Giugno 1999

F. Spagna

INTERFACCE GRAFICHE UTENTE AWT E DI NUOVA GENERAZIONE IN JAVA

F. Spagna

INTERACCE GRAFICHE UTENTE
AWT E DI NUOVA GENERAZIONE IN JAVA

1. Componenti grafici AWT ed eventi

1.1 Componenti grafici

1.1.1 Librerie AWT e Java Foundation Classes (JFC)

Le interfacce grafiche utente (*Graphical User Interface*, brevemente *GUI*) per le applet e applicazioni Java sono gestite dalla libreria di classi **Abstract Windowing Toolkit** (**AWT**), contenuta nel package `java.awt`, con la quale si possono creare delle interfacce a finestre con grafica, componenti di interazione con l'utente e gestire gli eventi relativi. Questa libreria è generalmente considerata ancora un po' rudimentale, ma nelle versioni più recenti del JDK (con il JDK 1.1) è stata introdotta come libreria di sistema (compresa cioè nel *core* di Java), la libreria di classi chiamata **Java Foundation Classes** (**JFC**), creata da Sun (Javasoft) in collaborazione con Netscape ed IBM, totalmente portabile sulle varie piattaforme, che estende la AWT con un insieme di componenti grafici ad alto livello e servizi avanzati, che rappresentano un miglioramento per lo sviluppo delle interfacce grafiche utente (GUI) rispetto all'AWT del JDK 1.0 da vari punti di vista e offrono tra l'altro migliori prestazioni. Questa libreria, che spinge più avanti la frontiera di Java verso applicazioni sofisticate e di livello commerciale, riduce le differenze di aspetto e di comportamento dell'interfaccia grafica che si possono manifestare con l'AWT sulle varie piattaforme. JFC contiene un modello di eventi per delega (con eventi specifici per ogni classe, che vengono inviati direttamente all'oggetto interessato), la possibilità di stampa e di trasferimento di dati con il *clipboard* (operazioni di "taglia e cuci"), miglioramenti per la grafica e per le immagini, l'introduzione di operazioni senza uso del mouse, menù di tipo *popup*, contenitori a pannello scorrevole e diverse altre possibilità, oltre al supporto dei JavaBeans [7.1].

[7.1] paragrafo tratto da: JFC – Documentation, java.sun.com/products/jfc/docs.html

[7.2] Java Foundation Classes, Now and the Future, White Paper, Sun.

Dal canto suo Microsoft ha sviluppato il Software Development Kit (SDK) per Java che comprende le classi AWT del JDK di JavaSoft, ma contiene in più una libreria di classi chiamate **Application Foundation Classes** (**AFC**) con funzionalità rivolte ai JavaBeans, alla sicurezza e all'accesso diretto alle API Win32 di Windows con J/Direct per la creazione di interfacce utente [7.3].

[7.3] tratto da PCWeek Italia, 22 set 97, pag. 57

1.1.2 Costituzione di un'interfaccia grafica

L'interfaccia grafica di un'applicazione Java o di un'applet è in generale costituita da un componente grafico principale, che funziona da contenitore ed è di tipo derivato da `Frame` per un'applicazione e da `Panel` per un'applet, essendo ambedue queste classi derivate dalla classe `Container`, cioè capaci di contenere altri componenti, ciascuno dei quali può funzionare a sua volta come contenitore di altri componenti e così via in una gerarchia di contenitori e contenuti in cui ciascun componente è contenuto da un altro e può contenerne a sua volta altri. La posizione in questa gerarchia determina diverse cose:

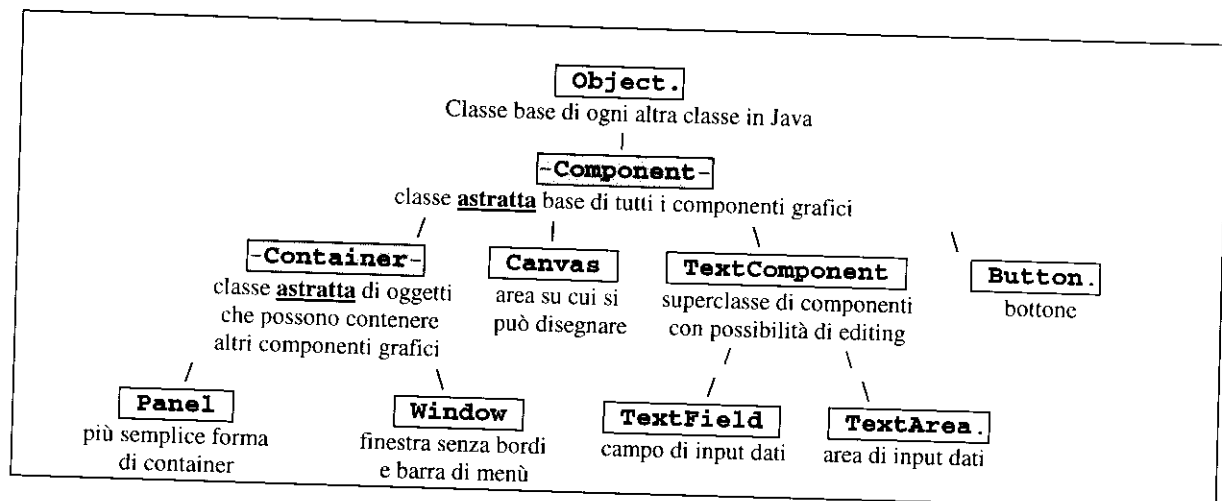
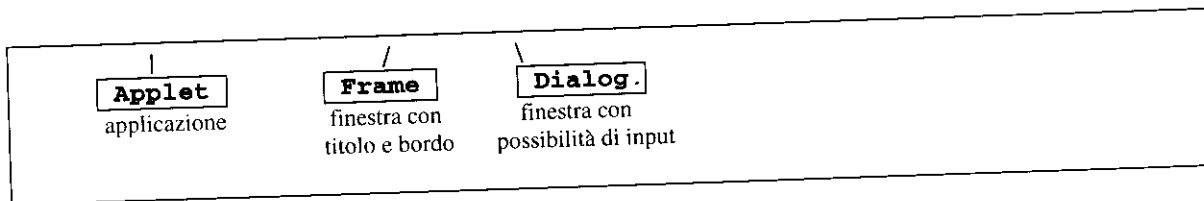
- il posizionamento dei componenti in assoluto sullo schermo che è definito in relazione al loro contenitore,

- l'ordine di tempo con il quale gli elementi sono disegnati ogni volta che l'applicazione viene rappresentata sullo schermo (cominciando da quelli più esterni fino a quelli più interni),

- il passaggio degli eventi da un componente a quelli superiori.

1.1.3 Gerarchia dei componenti grafici dell'AWT

Lo schema sottostante mostra la gerarchia di derivazione delle principali classi dell'AWT. Alla sommità della gerarchia, capostipite di ogni altra classe dell'AWT, sta la classe **`Component`**, che è astratta, e quindi non istanziabile direttamente, ma fornisce le funzionalità grafiche e di trattamento degli eventi di base di ogni componente grafico posto ovunque nella gerarchia di classi derivate. Tra le classi derivate possiamo distinguere da un lato i **contenitori** (oggetti derivati dalla classe **`Container`**, anch'essa astratta, tra i quali `Panel`, `Window`, `Frame` e `Dialog`), e dall'altro i componenti individuali come il `Canvas` e quelli di interazione con l'utente, come bottoni. menù di scelta, campi o aree di input (cioè `Button`, `Choice`, `TextField` e `TextArea`), che d'ora in poi chiameremo per intenderci **componenti UI** (dal termine *User Interface*) con il quale esse sono a volte conosciute.

Applet	Frame	Dialog.
applicazione	finestra con titolo e bordo	finestra con possibilità di input

1.1.4 La classe `Component` capostipite di tutti i componenti grafici

Poiché la classe astratta (che esiste cioè solo come progenitrice di altre classi, ma che di per sé non può essere istanziata direttamente) **Component** è, come si è detto, la classe base capostipite di tutte le classi di componenti grafici, è importante conoscerne subito almeno i metodi principali, che sono presenti per eredità in tutti gli altri componenti e conferiscono ad essi un minimo di funzionalità e comportamento comune. Quando si parlerà, tanto per fare un esempio, di un metodo `paint()` o `setColor()` della classe `Applet` bisogna tener presente che gli stessi metodi ci sono anche in una classe come la `Button` per esempio.

Facciamo qui di seguito un elenco parziale dei metodi principali, ma si fa presente che i metodi sono molti di più (nell'elenco sono stati tralasciati, per alleggerire il testo, gli argomenti (o parametri) richiesti dai metodi, che si vedranno invece in dettaglio quando si parlerà di ciascuno di essi via via che se ne presenterà l'occasione):

show()	rende visibile il componente
hide()	nasconde il componente
setForeground()	stabilisce il colore di disegno sul componente
getForeground()	restituisce il colore di disegno sul componente
setBackground()	stabilisce il colore di fondo sul componente
getBackground()	resituisce il colore di fondo sul componente
setFont()	stabilisce il font da utilizzare nelle scritte sul componente
getFont()	restituisce il font utilizzato nelle scritte sul componente
size()	restituisce le dimensioni del componente
setSize()	stabilisce le dimensioni del componente
getSize()	restituisce le dimensioni del componente
resize()	stabilisce le dimensioni del componente
reshape()	posiziona sullo schermo e dimensiona il componente in pixel
getGraphics()	crea e restituisce il contesto grafico tipo `Graphics` per il componente
getFontMetrics()	restituisce la metrica di un certo font
setCursor()	stabilisce il tipo di immagine del cursore

paint(Graphics)	disegna il componente sullo schermo
update()	aggiorna il disegno del componente
repaint()	ridisegna il componente
createImage()	crea un'immagine
inside()	verifica se il componente contiene un determinato punto

1.1.5 Definizione delle caratteristiche grafiche su un componente

Le caratteristiche grafiche che si applicano quando si disegna o si scrive su un componente possono essere definite in due modi diversi, riferendosi al componente stesso oppure al suo contesto grafico:

Esse possono essere fissate come caratteristiche grafiche proprie del componente che, come oggetto di una classe derivata dalla classe `Component`, dispone a questo riguardo dei metodi:

```
setBackground(Color)
setForeground(Color)
setColor(Color)
setFont(Font)
```

Le caratteristiche così fissate si applicano ad ogni cosa che viene disegnata o viene scritta sul componente, per esempio se in esso è disposta un'etichetta (`Label`) con una certa scritta, questa sarà fatta con il colore e il font fissati in questo modo.

Esempi#

Ma le caratteristiche grafiche possono anche essere imposte come caratteristiche del contesto grafico di tipo `Graphics` associato al componente, con cui si può disegnare o scrivere (oggetto ottenuto con il metodo `getGraphics()`, oppure passato automaticamente al `paint()` dal browser):

```
g.setColor(Color)
g.setFont(Font)
```

Ci si può allora chiedere quale dei due sistemi usare preferibilmente. E' da tener presente che le imposizioni fatte con il primo sistema prevalgono su quelli del secondo sistema.

vedi Esempio#

1.2 Componenti UI di interazione con l'utente

1.2.1 I componenti di interazione

I componenti grafici, tutti sempre derivati dalla classe `Component`, che l'AWT fornisce già preconfezionati al programmatore senza che egli debba preoccuparsi del loro disegno, e che costituiscono gli elementi di base per l'interazione con l'utente di un'interfaccia grafica, possono essere di tipo: `Label`, `Button`, `Checkbox`, `List`, `Choice`, `TextField`, `TextArea`, `Scrollbar`, `Menu`, `Canvas`.

Questi componenti, che noi chiameremo **componenti UI** (*User Interface*) in quanto sono utilizzati per l'interazione dell'applicazione con l'utente, devono essere contenuti da un contenitore (cioè da un oggetto di classe derivata dalla classe `Container`). Per inserire uno di essi in un contenitore (ad esempio in un `Panel` o in un'`Applet`) lo si istanzia prima come oggetto e lo si passa quindi al metodo **`add(Component)`** del contenitore (`add()` è un metodo di cui dispongono la classe `Container` e le sue derivate), ad esempio del `Panel`, quest'operazione essendo fatta di solito nel costruttore (o anche nel metodo `init()` se si tratta di un'applet). Ad esempio:

```
Button bot = new Button("OK");
add(bot);
```

oppure, con un'unica istruzione:

```
add(new Button("OK"));
```

ma in quest'ultimo modo non si ha a disposizione per il programma la referenza al bottone, con conseguenti limiti nell'uso.

1.2.2 Componente `Label`

Un **`Label`** (in inglese vuol dire etichetta e questo nome deriva dal suo frequente uso come etichetta per altri componenti) è costituita da un'area rettangolare contenente una stringa scritta sul componente che la contiene (di tipo `Container` o classe derivata).

Il risultato grafico è lo stesso di quello che si può ottenere con il metodo `drawString()` di `Graphics` (vedi paragrafo 7.4.8), e infatti i due modi possono essere usati in alternativa per scrivere qualcosa su un contenitore e per certe cose, secondo le circostanze, il `Label` può avere dei vantaggi da vari punti di vista:

- il suo disegno è automatico ogni volta che il contenitore viene ridisegnato e non richiede un'operazione specifica definita nel `paint()`,

- il suo posizionamento può essere automatico utilizzando un certo layout predefinito per il contenitore e non richiede così il calcolo dei pixel delle coordinate del punto di inizio, necessario

invece con un `drawString()` (per un posizinamento preciso a livello di pixel il `Label` dispone comunque, come oggetto derivato da `Component`, del metodo `reshape()`),

- le scritte possono essere facilmente allineate (a sinistra, al centro o a destra) rispetto al contenitore.

Lo svantaggio è che la scritta di un `Label` è opaca in quanto il fondo rettangolare dell'etichetta copre il disegno sottostante e non è disponibile per disegnarvi sopra altro oltre alla sua stringa di testo. Tutte queste considerazioni devono essere tenute presenti quando, dovendo scrivere con una stringa qualcosa in un certo posto di un contenitore, si prende in considerazione l'alternativa dei due modi diversi per farlo.

Esempio con i due modi:#

I costruttori possibili sono:

`Label()`	crea un'etichetta con stringa ancora nulla che potrà essere successivamente definita con il metodo `setText(String)`
`Label(String)`	crea un'etichetta inizializzata con stringa data come argomento e con allineamento di default a sinistra
`Label(String, int)`	crea un'etichetta con stringa data e con un altro argomento per definirne l'allineamento sul contenitore, usando le variabili di classe `Label.CENTER`, `Label.LEFT` e `Label.RIGHT`, o i loro valori (rispettivamente 0, 1 e 2)

Metodi principali:

`setText(String)`	assegna all'etichetta il valore della stringa
`getText()`	restituisce il valore della stringa dell'etichetta
`setAlignment(int)`	stabilisce il tipo di allineamento (per il valore dell'argomento vedi sopra)
`getAlignment()`	restituisce il tipo di allineamento (per i valori restituiti vedi sopra)

Esempio con tre righe aventi tre diversi allineamenti e una riga con due stringhe.#

Il risultato è riportato in figura 8.2.

Figura 8.2 Vari `Label` con diversi allineamenti.

Facciamo a questo punto un esempio di vari `Label` inseriti su un'applet: siccome l'aggiunta di un'etichetta è un'operazione molto semplice si è fatto qualcosa in più dell'inserimento di una sola etichetta mettendone una serie in un componente con una disposizione tipo `GridLayout` (vedi paragrafo 7.3.14) in una tabella che rappresenta una tavola pitagorica.

```
// H01label.java (F.Spagna) Esempio di Label disposti su una griglia

import java.awt.*;

public class H01label extends java.applet.Applet {

   public void init() {
      setFont(new Font("Courier", Font.BOLD, 16));          // fissa il carattere
      setBackground(Color.yellow);                            // colore di fondo
      setForeground(Color.blue);                          // colore delle scritte
      setLayout(new GridLayout(6, 10));                 // disposizione a griglia
      for (int r = 1; r <= 6; r++)                                 // per 6 righe
         for (int c = 1; c <= 10; c++)                          // per 10 colonne
            add(new Label("" + r*c));    // etichette con prodotto riga*colonna
   }
}
```

Il risultato è riportato nella figura 8.3.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60

Figura 8.3 Vari `Label` disposti in una griglia.

Ma nell'esempio precedente non viene dato un nome alle etichette, in quanto esse sono istanziate senza assegnarne una referenza, ma passando direttamente il `new` al metodo `add()` dell'applet. Può invece essere più utile poter disporre di un array di `Label` con un loro nome e ciò permetterebbe di poterne avere un controllo e cambiare la loro scritta o altre caratteristiche in qualunque momento nel corso di un programma. Segue un esempio in cui viene fatto questo che può essere visto come un esempio anche per qualsiasi altro tipo di componente, come una serie di

Button o di Checkbox. Si noti che l'istanziazione con il primo new dell'array crea un array di referenze agli oggetti, ma non crea ancora gli oggetti stessi, che sono invece creati con il for di new successivo.

Esempio:#

1.2.3 Componente **Button**

La classe **Button** rappresenta un bottone che può essere utilizzato dall'utente per far partire un'azione.

I costruttori possibili sono:

Button()	crea un bottone senza nessuna scritta, che potrà essere poi aggiunta con il metodo setLabel(String)
Button(String)	crea un bottone con una determinata scritta

Metodi principali:

setLabel(String)	assegna una scritta al bottone
getLabel()	restituisce il valore della scritta del bottone

Diversi esempi con bottoni sono stati fatti in questo testo a proposito dei vari gestori di layout e si rimanda ad essi (vedi paragrafo 7.3.12 e seguenti).

Si fa qui un altro esempio di vari bottoni con diverse scritte, dove si vede che la larghezza del bottone è proporzionata alla lunghezza della scritta che esso deve contenere.

Esempio:#

Figura 8.3 Vari Button con larghezze diverse in relazione alla loro scritta.

1.2.4 Componente **Checkbox**

Le **Checkbox** sono dei componenti che permettono all'utente di fare una scelta tra due opzioni attivandone una o disattivandola, così che il programma potrà prevedere dei comportamenti diversi in base allo stato del componente.

I costruttori possibili sono:

Checkbox()	crea una `Checkbox` non attivata
Checkbox(String)	crea una `Checkbox` affiancata a destra da un'etichetta con stringa data
Checkbox(String,null,boolean)	crea una `Checkbox`: l'ultimo argomento stabilisce se essa è inizialmente attivata o no; il secondo argomento, che deve essere `null` per le `Checkbox` singole, può essere utilizzato per poterle riunire in gruppi assegnando ad esso la referenza dell'eventuale gruppo, come vedremo al paragrafo seguente

Metodi principali:

setLabel(String)	assegna il testo dell'etichetta che affianca la `Checkbox`
getLabel()	restituisce il valore della stringa dell'etichetta che affianca la `Checkbox`
setState(boolean)	stabilisce lo stato della `Checkbox` (cioè se è attivata o no)
getState()	restituisce lo stato della `Checkbox` (`true` se attivata e `false` se no)

La classe `Checkbox`, in relazione ad un eventuale gruppo in cui fosse inserita, dispone dei due metodi:

setCheckboxgroup(Checkboxgroup g)	assegna una `Checkbox` ad un dato gruppo
getCheckboxgroup()	restituisce il gruppo cui appartiene una `Checkbox`

Si fa qui un esempio di due `Checkbox` inserite in un'applet, una inizialmente attivata e l'altra no.

```
// H02checkbox.java (F.Spagna) Esempio di Checkbox
```

```
import java.awt.*;

public class H02checkbox extends java.applet.Applet {

   Checkbox chb1, chb2;
   public void init() {
      chb1 = new Checkbox("prima");                                // Checkbox normale
      chb2 = new Checkbox("seconda", null, true); //Checkbox iniz.selezionata
      add(chb1);
      add(chb2);
   }
}
```

Il risultato è riportato in figura 8.4. Si noti la posizione assunta dai due componenti, che seguono la disposizione tipo `FlowLayout` per default e quindi sono situati sulla prima riga in alto e al centro.

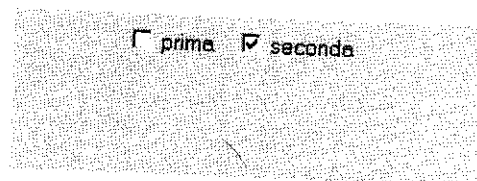



Figura 8.4 Due `Checkbox`, una disattivata ed una attivata.

1.2.5 Gruppi di **`Checkbox`** e classe **`CheckboxGroup`**

I vari componenti tipo `Checkbox` presenti in un contenitore possono essere tra di loro indipendenti oppure riuniti (indipendentemente dalla loro posizione, ma solo logicamente) in gruppi nei quali vale la regola che una sola `Checkbox` alla volta può essere attivata e tutte le altre sono allora disattivate, cioè ogni volta che si fa una scelta questa è **esclusiva** e disattiva la `Checkbox` precedentemente attivata (qualcosa come i *radio buttons* di Windows). E' la classe **`CheckboxGroup`** che permette di collegare insieme vari componenti `Checkbox` in un gruppo. Per creare un gruppo si istanzia un oggetto di questa classe e poi si usa la referenza a quell'oggetto come secondo argomento nel costruttore quando si creano le varie singole istanze di `Checkbox` (si ricordi il terzo costruttore visto al paragrafo precedente per `Checkbox`).

Costruttore:

`Checkboxgroup()`	crea un gruppo di `Checkbox`

Ecco un esempio in cui tre `Checkbox` sono riunite in un gruppo:

```
// H03checkboxgroup.java (F.Spagna) Esempio di Checkboxgroup
import java.awt.*;

public class H03checkboxgroup extends java.applet.Applet {

   Checkbox chb1, chb2, chb3;
   public void init() {
      CheckboxGroup grup = new CheckboxGroup();
      chb1 = new Checkbox("prima", grup, false);
      chb2 = new Checkbox("seconda", grup, false);
      chb3 = new Checkbox("terza", grup, false);
      add(chb1);
      add(chb2);
      add(chb3);
   }
}
```

E il risultato è presentato in figura 8.5.

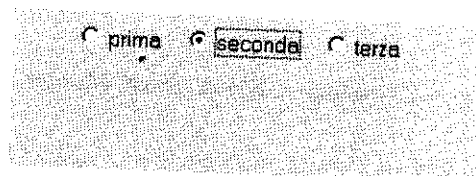



Figura 8.5 Tre `Checkbox` riunite in un `Checkboxgroup`.

1.2.6 Menù di tipo `Choice`

I componenti di tipo **`Choice`** (che in inglese vuol dire scelta) sono dei menù a tendina (menù *popup* o *pulldown*) che permettono di selezionare una tra varie voci (le voci sono dette ***item***).

Un componente di tipo `Choice` viene creato istanziando un oggetto della classe `Choice` e aggiungendovi poi, nell'ordine previsto, le varie voci (item) con il metodo **`addItem(String)`**.

Costruttore:

`Choice()`	crea un menù di tipo `Choice` senza ancora alcun item

Per andare a vedere nel programma il dato selezionato e per preselezionare un item la classe `Choice` mette a disposizione i metodi:

getSelectedIndex()	che restituisce l'indice dell'item scelto (il primo ha indice 0)
getSelectedItem()	che restituisce la stringa dell'item scelto
select(int)	che preseleziona un item con il suo indice (per il primo l'indice 0)
select(String)	che preseleziona un item mediante la sua stringa

Altri metodi della classe:

getItem(int)	che restituisce la stringa dell'item che occupa una certa posizione
countItems()	che restituisce il numero totale di item nel menù

Esempio di un menù di questo tipo con tre voci, inserito in un'applet:

```
// H04choice.java (F.Spagna) Esempio di menu' tipo Choice

import java.awt.*;

public class H04choice extends java.applet.Applet {

   Choice ch;
   public void init() {
      ch = new Choice();
      ch.addItem("prima voce");
      ch.addItem("seconda voce");
      ch.addItem("terza voce");
      ch.addItem("quarta voce");
      add(ch);
   }
}
```

In figura 8.6 viene mostrato come si presenta il menù dell'esempio in due situazioni diverse: all'inizio subito dopo la creazione e quando viene aperto dall'utente per selezionare una voce.

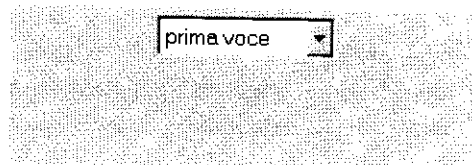
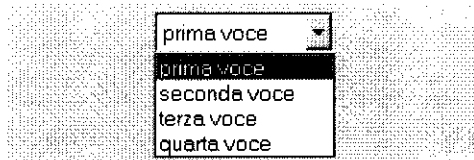



Figura 8.6 Menù di tipo `Choice` in condizioni di riposo e quando è aperto.

Altro esempio (con trattamento degli eventi):

```
// H05choice.java (F.Spagna) Esempio di menu' tipo Choice

import java.awt.*;

public class H05choice extends java.applet.Applet {

   Choice ch = new Choice();
   Label lab = new Label("nessuna scelta   ");
   String s[] = { "prima", "seconda", "terza" };
   public void init() {
      add(ch);
      for (int n = 0; n < 3; n++)
         ch.addItem(s[n]);
      add(lab);
  }
  public boolean action(Event evt, Object arg) {
     if (evt.target == ch)
        lab.setText(ch.getSelectedItem() + " scelta = "
                 + ch.getSelectedIndex());
     return true;
  }
}
```

1.2.7 Liste scorrevoli (classe **`List`**)

Una **`List`** è costituita da una serie di stringhe scorrevoli (*scrolling*) che assomiglia ad un componente di tipo `Choice` nel senso che permette una scelta da parte dell'utente tra varie voci, ma con le seguenti differenze:

- non si aprono come fanno invece i `Choice` solo su richiesta dell'utente, ma sono mostrate sempre con un certo numero predeterminato di righe e possono essere scorse con delle barre di scorrimento quando il numero delle voci è maggiore del numero di righe contemporaneamente visibili sullo schermo;

- può essere fatta dall'utente una selezione sia su una singola voce sia su più voci contemporaneamente, essendo questa, se una o più, una decisione che viene presa dal programmatore al momento della loro creazione (con il secondo costruttore dell'elenco che segue).

I costruttori sono:

`List()`	crea una `List` vuota
`List(int, boolean)`	questo costruttore richiede un argomento di tipo intero che rappresenta il numero di righe visibili sullo schermo ed un booleano che determina se sono permesse selezioni multiple oppure no

Il metodo **`addItem(String)`**, analogo a quello di `Choice`, permette l'aggiunta di nuove stringhe alla `List`.

Esempio di `List` in un'applet, molto simile a quello fatto per il `Choice` nel paragrafo precedente:

```
// H06list.java (F.Spagna) Esempio di List
import java.awt.*;

public class H06list extends java.applet.Applet {
   List lis;
   public void init() {
      lis = new List();
      lis.addItem("prima voce");
      lis.addItem("seconda voce");
      lis.addItem("terza voce");
      lis.addItem("quarta voce");
      lis.addItem("quinta voce");
      add(lis);
   }
}
```

La `List` prodotta dal codice dell'esempio è raffigurata in figura 8.7.

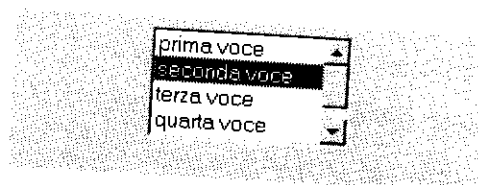



Figura 8.7 Un esempio di `List`.

1.2.8 Componente `TextField`

Un **`TextField`** è un campo di testo editabile che permette all'utente di introdurre da tastiera una serie di caratteri (compresi anche eventuali spazi) che possono essere letti dal programma sotto forma di stringa.

La stringa contenuta in un `TextField` può avere più caratteri di quelli che entrano nel campo visibile, in tal caso ci si può muovere lo stesso all'interno della stringa con i comandi del suo editore di testo interno, anche se non si può vedere la stringa tutta insieme nello stesso tempo, ma la si può fare scorrere all'interno del campo.

E' possibile far sì che i caratteri che l'utente inserisce da tastiera, pur correttamente introdotti nella stringa del `TextField`, non siano però visibili sullo schermo, ma siano mascherati da un carattere di eco, e questo è utile di solito per l'inserimento di password. Ciò si ottiene con il metodo `setEchoCharacter(char)` di questa classe, avente per argomento il carattere di eco sullo schermo che nasconde i caratteri reali inseriti, per esempio tipicamente:

```
setEchoCharacter("*");
```

I costruttori possibili sono:

`TextField()`	crea un `TextField` vuoto di larghezza nulla (non visibile)
`TextField(int)`	crea un `TextField` vuoto ma di larghezza visibile assegnata
`TextField(String)`	crea un `TextField` contenente una stringa preassegnata
`TextField(String,int)`	crea un `TextField` di larghezza data e con stringa assegnata

Metodi principali, ereditati dalla classe genitrice `TextComponent`, e quindi posseduti anche dalla classe sorella `TextArea` (vedi paragrafo seguente):

`setText(String)`	prefissa un testo di partenza (di default) nel campo di input
`getText()`	restituisce la stringa contenuta nel campo di input
`getColumns()`	restituisce la larghezza del campo in numero di caratteri
`select(int,int)`	seleziona il testo compreso tra due caratteri (con i 2 indici)
`selectAll()`	seleziona tutto il testo nel campo
`setEditable(boolean)`	con un argomento `false` rende il campo non editabile
`isEditable()`	restituisce `true` se il testo è editabile
`setEchoCharacter(char)`	impone un carattere di eco che maschera l'input dell'utente
`echoCharIsSet()`	dice se l'input è mascherato da un carattere di eco
`getEchoChar()`	restituisce il carattere di eco eventualmente adoperato

Esempio di un `TextField` con una larghezza di 20 caratteri inserito in un'applet:

```
// H07textfield.java (F.Spagna) Esempio di TextField

import java.awt.*;

public class H07textfield extends java.applet.Applet {

   TextField tf;
   public void init() {
      tf = new TextField(20);
      add(tf);
   }
}
```

In figura 8.8 è mostrato l'aspetto del `TextField` dell'esempio mentre l'utente sta introducendo dei caratteri (si può notare il cursore in fondo).

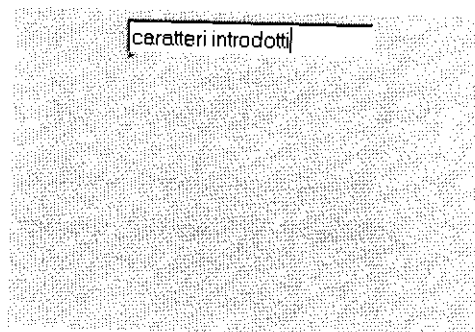



Figura 8.8 Un `TextField` durante l'introduzione dei dati.

1.2.9 Componente **TextArea**

Una **TextArea** è un'area editabile che permette l'inserimento dei caratteri di un testo da parte dell'utente (o da parte del programma, e in tal caso serve piuttosto per presentare un testo sullo schermo in un riquadro scorrevole) similmente ad un `TextField`, ma con la possibilità di trattare testi di più grandi dimensioni distribuiti su più righe visibili su un'area di larghezza ed altezza prefissate dal programmatore, con una barra di scorrimento verticale che permette di scorrere verticalmente e orizzontalmente il testo quando esso ha un numero di righe maggiori di quelle contemporaneamente visibili sullo schermo.

I costruttori possono essere:

TextArea()	crea una `TextArea` vuota di dimensioni non definite (non visibile)

TextArea(int,int)	crea una `TextArea` vuota con numero di righe e di colonne dati
TextArea(String)	crea una `TextArea` senza dimensioni contenente un testo dato
TextArea(String,int,int)	crea una `TextArea` contenente un testo dato e con numero di righe e di colonne dati

Essendo `TextArea` una classe derivata dalla classe `TextComponent` come `TextField`, condivide con `TextField` diversi metodi di quella classe madre. Ma questa classe ha anche suoi metodi specifici:

getColumns()	restituisce la larghezza dell'area di testo (in numero di caratteri)
getRows()	restituisce il numero di righe visibili sullo schermo
insertText(String,int)	inserisce una stringa nel testo a partire da una posizione data
replaceText(String,int,int)	sostituisce il testo compreso tra due caratteri con un altro testo

Esempio di una `TextArea` inserita in un'applet:

```
// H08textarea.java (F.Spagna) Esempio di TextArea

import java.awt.*;

public class H08textarea extends java.applet.Applet {

   TextArea ta;
   public void init() {
      ta = new TextArea(6, 15);
      add(ta);
   }
}
```

metterci del testo in un'altra che espone testo dato con setEditable(false)#

In figura 8.9 si vede l'aspetto assunto sull'applet dalla `TextArea` dopo che l'utente ha introdotto un certo numero di righe di testo (si può notare il cursore che è ancora alla fine dell'ultima riga).

Figura 8.9 Un esempio di `TextArea`.

1.2.10 Barre di scorrimento (classe **`Scrollbar`**)

Le **`Scrollbar`** (barre di scorrimento) sono componenti grafici che permettono ad un utente di scegliere manualmente, agendo su un cursore, un certo valore compreso tra un minimo ed un massimo prestabiliti.

Come è ben noto ad ogni utilizzatore di un sistema grafico a finestre, sia esso di tipo Windows, Macintosh o Motif (Unix), l'utente può fare con il mouse delle operazioni su una barra di scorrimento (equivalenti a variazioni della grandezza rappresentata dalla `Scrollbar`) in tre modi diversi: agendo sulle frecce poste alle estremità della barra per piccoli spostamenti del cursore, sulle zone della barra interne superiore o inferiore al cursore per spostamenti più grandi (una pagina), o infine trascinando direttamente il cursore (chiamato a volte anche "ascensore") per spostamenti liberi.

Disegno#:

I costruttori disponibili sono:

`Scrollbar()`	crea una barra verticale con minimo e massimo uguali a zero
`Scrollbar(int)`	crea una barra verticale o orizzontale in relazione all'argomento che si può fissare con una delle due variabili di classe di tipo intero `Scrollbar.VERTICAL` e `Scrollbar.HORIZONTAL` (rispettivamente di valore 0 e 1)
`Scrollbar(int,int,int,int,int)`	avente cinque argomenti interi: il primo stabilisce se la barra è verticale o orizzontale (vedi sopra) il secondo dà il valore iniziale (compreso tra minimo e massimo) il terzo assegna la lunghezza della barra (in ...#) il quarto fissa il valore minimo il quinto fissa il valore massimo

Metodi principali:

`getMinimum()`	restituisce il valore minimo
`getMaximum()`	restituisce il valore massimo
`getOrientation()`	restituisce l'orientamento (0 se verticale e 1 se orizzontale)
`getValue()`	restituisce il valore corrente determinato dalla posizione del cursore
`setValue()`	stabilisce il valore corrente e posiziona il cursore di conseguenza

Di seguito viene riportato un esempio con una barra orizzontale e valori compresi tra 0 e 100. E' da tenere presente che le dimensioni della barra dipendono dal *layout manager* attivo (se il *layout manager* non è fissato la barra prende un aspetto anomalo): se per esempio si fissa un BorderLayout e si aggiunge una barra orizzontale a nord si ottiene una barra che corre lungo tutto il bordo superiore del contenitore, come nell'esempio.

```
// H09scrollbar.java (F.Spagna) Esempio di Scrollbar

import java.awt.*;

public class H09scrollbar extends java.applet.Applet {

   Scrollbar sbar;
   Label lab;
   public void init() {
      setLayout(new BorderLayout());
      sbar = new Scrollbar(Scrollbar.HORIZONTAL, 50, 0, 0, 101);
      add("North", sbar);
      lab = new Label("" + sbar.getValue());
      lab.setFont(new Font("", Font.BOLD, 50));
      add("Center", lab);
   }
   public boolean handleEvent(Event e) {
      if (e.target == sbar) {
         lab.setText("" + sbar.getValue());
         return true;
      }
      return super.handleEvent(e);
   }
}
```

Il componente dell'esempio è mostrato in figura 8.10.

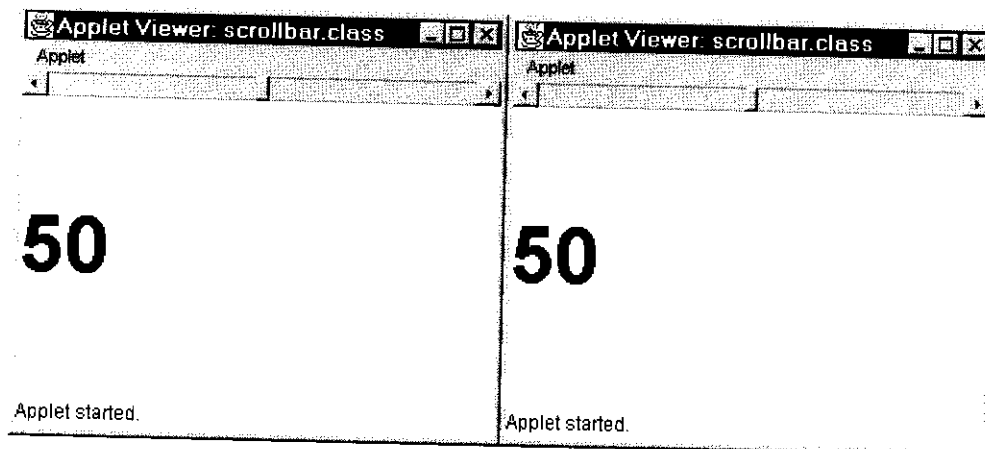



Figura 8.10 Esempio di `Scrollbar`.

fai due posizioni°

1.2.11 Componente `Canvas`

La classe **`Canvas`** rappresenta un componente che dispone di una superficie rettangolare vuota sulla quale si può disegnare con i metodi dell'oggetto di tipo `Graphics` che viene passato al suo metodo `paint()`, e può anche, in quanto derivata da `Component`, intercettare gli eventi, e in questi suoi aspetti fa quello che può fare un `Panel`, ma, diversamente da questo, non essendo essa derivata dalla classe `Container`, non può contenere altri componenti.

In generale viene creato un oggetto di una classe derivata da `Canvas` nella quale è ridefinito il metodo `paint()` e il metodo di risposta agli eventi, e quest'oggetto è aggiunto ad un contenitore con il metodo `add()`.

Un `Canvas` può essere utilizzato per esempio come supporto per un'immagine che può essere sistemato in un contenitore per mezzo di un gestore di layout (layout manager). Esempio:#

1.3 Contenitori di componenti grafici

1.3.1 Contenitori e classe **Container**

Le classi di contenitori grafici, derivate, come ogni altra classe AWT, dalla classe `Component`, rappresentano dei componenti che possono contenere degli altri componenti AWT e quindi anche degli altri contenitori, e sono derivate dalla classe **`Container`**, che, essendo astratta, non può essere istanziata direttamente, ma in cui è predisposta la capacità di funzionamento come contenitore. L'ordine di aggiunta dei componenti è tenuto presente per certi effetti (vedi paragrafo 7.#). I componenti contenuti sono rappresentati in un array, variabile d'istanza del contenitore, `Component components[]`.

Le classi dell'AWT di questo tipo sono `Panel` (e quindi anche `Applet`, che deriva da `Panel`), `Window`, `Frame` e `Dialog`.

Oltre a ricordare tutti i metodi ereditati dalla classe `Component` (vedi paragrafo 7.1.4), riportiamo qui alcuni metodi specifici della classe `Container`:

`add(Component)`	aggiunge un componente al contenitore
`remove(int)`	toglie un componente caratterizzato dal suo indice
`getComponentCount()`	restituisce il numero di componenti contenuti nel contenitore
`getComponent(int n)`	restituisce il componente di indice `n` del contenitore
`Component[] getComponents()`	restituisce l'array dei componenti del contenitore

1.3.2 Contenitori di classe **Panel**

La classe **`Panel`** rappresenta un contenitore di componenti (che possono essere anche altri `Panel`) nella sua forma più semplice.

Le dimensioni di un `Panel` (e quindi anche di un'`Applet`) possono essere conosciute attraverso il metodo `size()` che restituisce un oggetto di tipo `Dimension`, i cui membri `width` e `height` (cioè `size().width()` e `size().height()`) danno la larghezza e l'altezza del `Panel` o `Applet` in pixel.

Costruttori:

`Panel()`	crea un `Panel` con un layout di default di tipo `FlowLayout`
`Panel(LayoutManager)`	crea un `Panel` con un `LayoutManager` specificato

1.3.3 Oggetti di classe **Applet** come contenitori

Essendo la classe `Applet` una sottoclasse della classe `Panel`, tutto quello che si può dire per `Panel` come contenitore vale anche per `Applet`, in altre parole si potrebbe dire che un'applet è in fondo, sotto l'aspetto delle caratteristiche di contenitore, un `Panel`, anche se ha poi tante altre funzionalità in più. E questo anche se la classe `Applet`, per la sua specificità è contenuta in un suo package a parte (il package `java.applet`) separato da quello delle altre classi AWT.

1.3.4 Contenitori di classe **Window**

La classe **Window** permette la creazione di finestre indipendenti, da quella del browser in un'applet e dal `Frame` principale del programma in un'applicazione, alle quali possono essere aggiunti dei menù.

Questa classe fornisce la funzionalità di base di una finestra. Generalmente vengono usate le sue due sottoclassi `Frame` e `Dialog` (*dialog box*), che è un tipo di finestra più semplice.

1.3.5 Contenitori di classe **Frame**

La classe **Frame** rappresenta una finestra (è derivata da `Window`) con un titolo ed un bordo, che contiene in sé tutto il comportamento di una finestra e la possibilità di inserzione di una barra dei menù.

I costruttori sono:

Frame()	crea una finestra di tipo `Frame` senza titolo
Frame(String)	crea una finestra di tipo `Frame` con un titolo assegnato

Metodi:

setTitle(String)	permette di assegnare un titolo alla finestra
getTitle()	restituisce la stringa del titolo
setMenuBar(MenuBar)	aggiunge#
getMenuBar()	restituisce la referenza alla barra dei menù (oggetto `MenuBar`)
isResizable()	dice se la finestra è ridimensionabile
remove(MenuComponent)	toglie la barra dei menù dalla finestra
dispose()	rilascia tutte le risorse impegnate dalla finestra e dai suoi componenti

add()	per aggiungervi degli elementi grafici (ad esempio un Label per inserire una scritta), che Frame ha in quanto derivata della classe Container,
pack()	per proporzionare le dimensioni della finestra stessa in relazione ai componenti che essa contiene.

Ad un Frame si possono aggiungere dei componenti come in Panel con la funzione add(). Ma, contrariamente ai Panel in cui il gestore di layout (LayoutManager) di default era il FlowLayout, nei Frame il gestore di layout di default è il BorderLayout.

Le dimensioni di una finestra possono essere assegnate con il metodo resize() e la posizione del vertice in alto a sinistra sullo schermo può essere stabilita con il metodo move().

Una finestra creata diventa visibile solo con il metodo show(), posseduto da ogni componente (oggetti di classe derivata da Component), che ha il suo contrario nel metodo hide() che la fa invece scomparire dallo schermo.

Un Frame può produrre degli eventi propri come:

WindowOpened, WindowClosing, WindowClosed, WindowIconified,

WindowDeiconified, WindowActivated, WindowDeactivated.

Esempio di pag. 281 ma senza la 2a classe #

1.3.5.1 Creazione di una finestra

Per creare una finestra in un'applet o in un'applicazione Java (nel caso di un'applicazione si può trattare della stessa finestra principale che ospita l'applicazione o di una nuova finestra a sé stante) si istanzia un Frame e se ne invoca il metodo show() per renderla visibile.

Esempio di codice per la creazione di una finestra di tipo Frame:esempio completo#

```
Frame f = new Frame();        // crea una finestra
f.setTitle("titolo");         // le da' un titolo
f.add(new Label("scritta"); // vi aggiunge un componente (una scritta)
f.pack();                     // proporziona la finestra al contenuto
f.show();                     // rende visibile la finestra
```

1.3.5.2 Chiusura di una finestra tipo `Frame` o `Dialog`

Le finestre di classe `Frame` o `Dialog` non rispondono in generale al comando di chiusura tipico delle finestre perchè tale evento non è previsto nella loro classe: per attivare questa funzionalità si può sottoclassare da esse una finestra particolare e ridefinirne il metodo di trattamento degli eventi `handleEvent()` (ogni classe derivata da `Component` ne ha uno) nel modo seguente:

Esempio completo#

```
public boolean handleEvent(Event ev) {
   if (ev.id == Event.WINDOW_DESTROY)
      dispose();
   return super.handleEvent(ev);
}
```

Con il modello di eventi 1.1 una finestra tipo `Frame` chiudibile può essere creata con un codice come quello che segue:

```
// H00chiudib.java (F.Spagna) Finestra tipo Frame chiudibile

import java.awt.*;
import java.awt.event.*;

class FrameChiudibile extends Frame implements WindowListener {

   FrameChiudibile() {
      super("Frame chiudibile");
      addWindowListener(this);
   }
   public void windowClosing(WindowEvent e) { dispose(); }
   public void windowClosed(WindowEvent e) { }
   public void windowOpened(WindowEvent e) {  }
   public void windowActivated(WindowEvent e) {   }
   public void windowDeactivated(WindowEvent e) {   }
   public void windowIconified(WindowEvent e) {   }
   public void windowDeiconified(WindowEvent e) {   }
}

public class H00chiudib extends FrameChiudibile {

   public void paint(Graphics g) {
      g.drawString("finestra chiudibile", 20, 40);
   }
   public static void main(String s[]) {

      H00chiudib fr = new H00chiudib();
      fr.setSize(320, 200);
      fr.pack();
      fr.setVisible(true);
   }
}
```

1.3.6 Contenitori di classe `Dialog`

La classe **`Dialog`**, derivata dalla classe `Window`, permette la creazione di finestre tipo *dialog box*, più semplici di quelle di tipo `Frame`, che non possono avere dei menù e che sono di solito usate per mostrare avvisi all'utilizzatore o per ricevere da esso una risposta (un bottone OK su una `Dialog` la fa chiudere) o qualche dato (da cui il suo nome), anche se non ci sono limiti al loro uso come contenitori di componenti UI o per qualsiasi operazione grafica.

Una `Dialog` ha sempre una finestra madre (*parent*) di tipo `Frame` (una `Dialog` richiede necessariamente un `Frame` preesistente). Il costruttore di questa classe richiede infatti il nome della finestra *parent*: si utilizza allora il metodo `getParent()` che restituisce la referenza ad un oggetto di tipo `Container` e quindi per avere un `Frame` è richiesto un *casting*.(?#)

Come i `Frame`, anche le `Dialog` sono rese visibili solo con un metodo `show()`, mentre il metodo `hide()` le fa scomparire.

Costruttori:

`Dialog(Frame,boolean)`	crea una `Dialog` collegata ad un `Frame` con un argomento che stabilisce se essa è modale, cioè se richiede di essere chiusa prima che l'utilizzatore possa agire su altre finestre
`Dialog(Frame,String,boolean)`	come sopra, ma con una barra di titolo ed un titolo

Esempio#

1.3.7 Classe `ScrollPane`

La classe **`ScrollPane`** del package `java.awt` è una sottoclasse di `Container` che implementa automaticamente in un contenitore la funzionalità di fare scorrere (*scrolling*) orizzontalmente e verticalmente al suo interno un singolo componente contenuto in esso (che di seguito chiameremo ***"child"***, cioè figlio) mediante scrollbar. Per le scrollbar può essere fatta una scelta (*"display policy"*) tra le seguenti possibilità:

- le scrollbar sono mostrate solo quando sono necessarie (*"as needed"*)
- le scrollbar sono mostrate sempre (*"always"*)
- le scrollbar non sono mai presenti (*"never"*)

con un parametro fissato tra gli argomenti al momento della costruzione.

Lo stato relativo alla posizione corrente del componente scorrevole *child* è rappresentato da due oggetti (uno per ciascuna dimensione) che implementano l'interfaccia `Adjustable`. A questi oggetti si può accedere con metodi della classe per rilevare o modificare gli attributi (come unità di incremento, valore, etc.).

Alcune proprietà (minimo, massimo, e quantità visibile) sono fissate internamente dallo `ScrollPane` in relazione alla geometria dello `ScrollPane` e del suo *child* e in generale non dovrebbero essere fissate dal programma.

Anche nel caso in cui le scrollbar fossero definite come mai presenti ("never") il contenuto child dello scrollpane può essere fatto scorrere a programma usando il metodo `setScrollPosition()`. Questo si rende utile in particolare quando vengono usati nel programmma sistemi di controllo del movimento diversi dalle scrollbar (si veda l'esempio fatto in questo paragrafo).

Le dimensioni iniziali del contenitore sono poste ai valori 100 x 100, ma possono essere impostate diversamente con il metodo `setSize()`.

Per definire gli spazi tra le scrollbar e i bordi del contenitore si possono usare gli `Insets` (vedi paragrafo 7.3.17). Il valore attuale degli `Insets` può essere rilevato con il metodo `getInsets()` e può cambiare dinamicamente in modo automatico in dipendenza del fatto che le scrollbar siano al momento visibili oppure no.

1.3.7.1 Variabili di classe

Si fa qui un elenco delle variabili di classe, tutte `public final static int`:

SCROLLBARS_AS_NEEDED

Specifica che le scrollbar orizzontale e verticale sono mostrate solo quando la dimensine del componente scorrevole child supera quella dello scrollpane contenitore in direzione orizzontale o verticale.

SCROLLBARS_ALWAYS

Specifica che le scrollbar orizzontale e verticale sono mostrate sempre, indipendentemente dalle dimensioni del conteniyoe scrollpane e del componnete scorrevole child.

SCROLLBARS_NEVER

Specifica che le scrollbar orizzontale/verticale non siano mai mostrate.

1.3.7.2 Costruttori

`ScrollPane()`

Crea un nuovo contenitore scrollpane con una "display policy" per le scrollbar posta per default al valore di "as needed".

`ScrollPane(int scrollbarDisplayPolicy)`

Crea un nuovo contenitore scrollpane specificando con il suo argomento la "display policy" relativa a quando le scrollbar devono essere mostrate.

1.3.7.3 Metodi

`addImpl(Component comp, Object constraints, int index)`

Aggiunge un componente (comp) al contenitore scrollpane. Se lo scrollpane ha già un suo componente child, quel componente è rimosso ed è sostituito dal nuovo.
I parametri sono, oltre al componente da aggiungere, constraints che non è applicabile e index che si riferisce alla posizione del componente child (deve essere <= 0)
(ridefinisce il metodo addImpl() della classe Container).

`int getScrollbarDisplayPolicy()`

restituisce la "display policy" per la presenza delle scrollbar.

`Dimension getViewportSize()`

Restituisce le dimensioni attuali (in pixel) del view port dello scrollpane.

`int getHScrollbarHeight()`

Restituisce l'altezza (in pixel) che sarebbe occupata da una scrollbar orizzontale, independentemente dal fatto che essa sia al momento mostrata dallo scrollpane oppure no.

`int getVScrollbarWidth()`

Restituisce la larghezza (in pixel) che sarebbe occupata da una scrollbar verticale, independenteemente dal fatto che essa sia al momento mostrata dallo scrollpane oppure no.

`Adjustable getVAdjustable()`

Restituisce un oggetto Adjustable che rappresenta lo stato della scrollbar verticale. Se la "display policy" per le scrollbar è "never", questo metodo restituisce null.

`Adjustable getHAdjustable()`

Restituisce un oggetto Adjustable che rappresenta lo stato della scrollbar orizzontale. Se la "display policy" per le scrollbar è "never" questo metodo restituisce null.

`setScrollPosition(int x, int y)`

Produce uno scorrimento fino alla posizione specificata entro il contenitore del componente child (si veda l'esempio fatto in questo paragrafo). Se si specifica una posizione che è al di fuori dei limiti legali di scorrimento del child lo scorrimento avverrà fino alla posizione legale più vicina. I limiti legali sono definiti quelli del rettangolo: x = 0, y = 0, larghezza = (child width - view port

width), altezza = (child height - view port height). Questo metodo, come anche il successivo, accede agli oggetti `Adjustable` che rappresentano lo stato delle scrollbar.
I parametri sono la posizione x e la posizione y fino alle quali si fa lo scorrimento.

`setScrollPosition(Point p)`

Produce uno scorrimento fino alla posizione specificata entro il contenitore del componente child (si veda anche quanto detto per il metodo precedente).
L'argomento rappresenta la posizione del punto (oggetto di tipo `Point`) fino al quale si fa lo scorrimento.

`Point getScrollPosition()`

Restituisce le coordinate (x,y) della posizione corrente di scorrimento entro il child che è mostrata alla posizione (0,0) del view port del panel fatto scorrere. Questo metodo, come anche il successivo, accede agli oggetti `Adjustable` che rappresentano lo stato delle scrollbar.

`setLayout(LayoutManager mgr)`

Stabilisce il layout manager per il contenitore.
L'argomento rappresenta il layout manager specificato.
(ridefinisce in modo specifico il metodo `setLayout()` della classe `Container`: il metodo della superclasse è ridefinito come final per impedire che il layoutmanager sia manipolato.

`doLayout()`

Ridimensiona il componente scorrevole child alle dimensioni preferite dal contenitore (se in tal modo la posizione attuale di sorrimento risultasse invalida, essa è posta alla posizione valida più vicina).
(ridefinisce in modo specifico il metodo `doLayout()` della classe `Container`).

`printComponents(Graphics g)`

Stampa il componente nello `ScrollPane`.
Il parametro di tipo `Graphics` rappresenta la finestra specificata.
(ridefinisce il metodo `printComponents()` della classe `Container`).

`String paramString()`

Restituisce il parametro `String` del contenitore.
(ridefinisce il metodo `paramString()` della classe `Container`).

Viene qui fatto un esempio in cui un'immagine è disegnata su un `Canvas` che costituisce l'elemento scorrevole di uno `ScrollPane` inserito in un'applet. Lo scorrimento viene effettuato

con le scrollbar presenti, ma anche a programma in relazione agli spostamenti del cursore del mouse sopra un campione dell'immagine disegnata in un angolo dell'applet.

```
// H10scroll051.java (F.Spagna) Esempio di ScrollPane
// 01-05.10.98  (inizio 4 ottobre 1998)
import java.awt.*;

public class H10scroll051 extends java.applet.Applet {  // nostra applet
   scrolpan sp;
   Image img;                       // ScrollPane che l'applet conterra'
   public void init() {
      img = getImage(getDocumentBase(),              // inizializza l'applet
      sp = new scrolpan(ScrollPane.SCROLLBARS_ALWAYS, img);
      add(sp);
   }                          // aggiunge lo ScrollPane creato sopra
   public void paint(Graphics g) {    // come disegna il nostro Canvas
      g.drawImage(img, 0, 0, 33, 43, this);
   }                                     //...impicciolita sull'appler
   public boolean mouseMove(Event e, int x, int y) { // se drag mouse
      if (x < 32 && y < 43)
         sp.setScrollPosition(10*x, 10*y);
      return true;
   }
}                                        //...impicciolita sull'appler
class scrolpan extends ScrollPane {     // il nostro tipo di ScrollPane
   scrolpan(int i, Image img) {         // costruttore dello ScrollPane
      super(i);                         // costruttore della suoerclasse
      setSize(150, 150);                    // dimensiona lo ScrollPane
      addImpl(new tela(img), null, 0);//aggiunge un Canvas scorrevole
   }
}
class tela extends Canvas {
   Image img;                              // il nostro tipo di Canvas
   tela(Image img) {                   // immagine trattata dal Canvas
      setSize(331, 432);          // costruttore che richiede un'immagine
      this.img = img;                          // dimensiona il Canvas
   }                            // immagine passata al costruttore
   public void paint(Graphics g) {     // come disegna il nostro Canvas
      g.drawImage(img, 0, 0, this);   // disegna l'immagine sul Canvas
   }
}
```

In figura 8.11 è presentato il risultato del programma come visto dall'appletviewer.

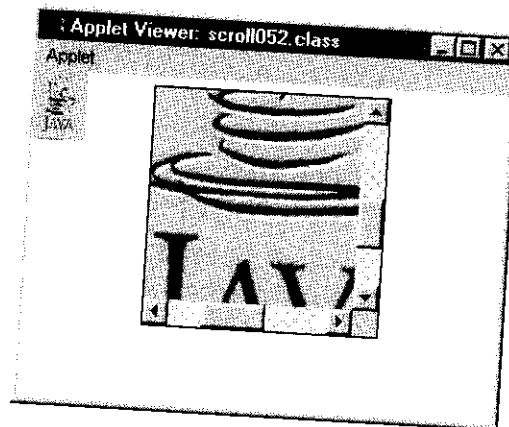



Figura 8.11 Esempio di un'applet con uno `ScrollPane` contenente un `Canvas` scorrevole.

1.3.8 Componenti **`Menu`** e **`Menubar`**

Le finestre tipo `Frame` possono avere in alto una barra dei menù contenente uno o più menù, ciascuno dei quali può presentare un certo numero di voci (***item***).

Una barra dei menù viene creata istanziando la classe **`MenuBar`** e può essere assegnata ad un `Frame` mediante il metodo `setMenuBar(MenuBar)` del `Frame`.

Ogni menù di una barra di menù può essere creata come istanza della classe **`Menu`** ed essere aggiunta alla barra con il metodo `add()` di questa.

Infine le singole voci di ogni menù sono aggiunte ad esso come oggetti della classe `MenuItem` mediante il metodo `add()` dei `Menu`.

Ecco la sequenza delle operazioni per un solo menù con tre voci:

#

Un `Menu` può essere disattivato con il metodo `disable()` della classe `Menu`, il cui contrario è il metodo `enable()` per riattivarlo.

In un menù possono essere aggiunti oltre che semplici voci anche dei sottomenù:

vedi esempio#

Un tipo speciale di voce (item) che si può aggiungere ad un menù è il **`CheckboxMenuItem`**, che è una voce contenente una casella selezionabile (checkbox). Questo componente, essendo una sottoclasse di `MenuItem`, viene trattato nello stesso modo di quello.

Vedi esempio#

Un altro elemento che si può aggiungere ad un menù è un separatore, costituito da una linea per separare diversi gruppi di voci del menù: esso viene introdotto come qualsiasi altra voce del menù, ma con una stringa di testo rappresentata da un semplice segno meno "-".

Come era possibile per i menù, anche le voci di menù possono essere disattivati e riattivati con i metodi `disable()` e `enable()` che anche la classe `MenuItem` possiede.

Esempio completo#

1.3.9 Finestra tipo **`FileDialog`**

La classe **`FileDialog`** permette la creazione di una finestra di accesso al sistema di file della macchina dell'utente (nel caso di applet il sistema di file è solo mostrato ma non si può agire su di esso perché, come si sa, per ragioni di sicurezza, le applet non possono agire sul file system). Tali finestre devono essere, come le `Dialog`, appoggiate ad un `Frame` preesistente.

La classe **`FileDialog`**, derivata da `Dialog`, rappresenta una finestra di selezione per l'apertura o il salvataggio di file. Si tratta di una dialog modale che blocca l'esecuzione dal momento in cui la finestra è mostrata con il metodo `show()` fintanto che non sia stata fatta una selezione sul file da parte dell'utente.

Variabili di classe:

`FileDialog.LOAD`	variable relativa al caricamento (load) di un file
`FileDialog.SAVE`	variable relativa al salvataggio (save) di un file

Costruttori:

`FileDialog(Frame)`	crea una `FileDialog` per aprire un file
`FileDialog(Frame,String)`	crea una `FileDialog` con un dato titolo per aprire un file
`FileDialog(Frame,String,int)`	crea una `FileDialog` con un dato titolo e modo potendo il modo essere posto come `FileDialog.LOAD` o `FileDialog.LOAD`

vedi#

FileDialog(Frame,boolean)	crea una `FileDialog` collegata ad un `Frame` con un titolo (etichetta di Load File)
FileDialog(Frame,String,boolean)	come sopra, ma con un argomento in più per scegliere il tipo di etichetta con una variabile di classe `FileDialog.LOAD` o `FileDialog.SAVE`

La scelta fatta dall'utente su una `FileDialog` può essere utilizzata nel programma servendosi dei metodi **getDirectory()** e **getFile()**, che possono servire per fare operazioni di input/output sui file scelti.

Metodi:

getDirectory(Frame,boolean)	restituisce la stringa del nome della directory selezionata
getFile(Frame,String,boolean)	restituisce la stringa del nome del file selezionato
getFilenameFilter()	restituisce il filtro
getMode()	restituisce il modo della `FileDialog`
paramString()	restituisce il parametro String della `FileDialog`
setDirectory(String)	stabilisce una directory
setFile(String)	stabilisce un file
setFilenameFilter(FilenameFilter)	stabilisce il filtro
setMode(int)	stabilisce il modo della `FileDialog`

Viene fatto di seguito un esempio di un'applicazione che presenta un menù per fare delle operazioni di caricamento o salvataggio di file mediante le relative versione della `FileDialog`.

```
// H11file061.java (F.Spagna) Esempio di FileDialog
// 01-18.12.98 (inizio:18 dic 98)

import java.awt.*;
import java.awt.event.*;

public class H11file061 extends Frame
                        implements ActionListener, WindowListener {
   String s[] = { "Apri", "Salva", "Chiudi" };
```

```
    TextArea ta = new TextArea();                         // area di testo editabile
    file062() {
       super("Editore di testo");                          // titolo della finestra
       add("Center", ta);                           // aggiunge l'area di testo
       MenuBar mb = new MenuBar();                                // barra dei menu'
       setMenuBar(mb);
       Menu m1 = new Menu("File");             // un solo menu' sulla barra dei menu'
       mb.add(m1);
       MenuItem it[] = new MenuItem[3];                // 3 item nel menu' singolo
       for (int n = 0; n < 3; n++) {                // per ciascuno dei 3 item...
          it[n] = new MenuItem(s[n]);                              // scritta item
          m1.add(it[n]);                                          // aggiunta item
          it[n].setActionCommand(s[n]);                           // comando item
          it[n].addActionListener(this);                          // listner item
       }
       setSize(250, 300);
       pack();
    }
    public void windowClosing     (WindowEvent ev) { System.exit(0); }
    public void windowClosed      (WindowEvent ev) { }
    public void windowDeiconified(WindowEvent ev) { }
    public void windowIconified   (WindowEvent ev) { }
    public void windowActivated   (WindowEvent ev) { }
    public void windowDeactivated(WindowEvent ev) { }
    public void windowOpened      (WindowEvent ev) { }

    public void actionPerformed  (ActionEvent ev) {
       if (ev.getActionCommand().equals("Apri")) {
          FileDialog f = new FileDialog(this, "Apri", FileDialog.LOAD);
          f.show();
          String fil = f.getFile();
          ta.setText(fil);
       }
       if (ev.getActionCommand().equals("Salva")) {
          FileDialog f = new FileDialog(this, "Salva", FileDialog.SAVE);
          f.show();
          String fil = f.getFile();
          ta.setText(fil);
       }
       if (ev.getActionCommand().equals("Chiudi"))
          System.exit(0);
    }
    public static void main(String args[]) {
       file061 ed = new file061();
       ed.show();
       ed.addWindowListener(ed);
    }
}
```

1.3.10 Classi *peer*

Le classi ***peer*** (package `java.awt.peer`) sono usate dall'AWT per realizzare i componenti sul sistema specifico relativo alla particolare piattaforma usata dall'utente corrispondenti alle classi AWT definite in un programma Java. Ogni volta che si crea un oggetto grafico dell'AWT viene contemporaneamente creato un oggetto *peer* corrispondente implementato in codice nativo e specifico della piattaforma, che si occupa di disegnare la grafica, ad esempio per un `Frame` viene

creata la finestra come `FramePeer`. C'è da dire che questa tecnica comporta una grande differenza di aspetto di un componente grafico sulle diverse piattaforme e anche qualche preoccupante differenza di comportamento, con conseguenti problemi di portabilità. Si vedrà al capitolo 8 come a questi inconvenienti ha posto infine rimedio la libreria di classi Swing introdotta nelle ultime versioni del JDK, anche se a costo di un ulteriore leggero scadimento delle prestazioni.

1.3.11 Posizionamento dei componenti in un contenitore e classe `LayoutManager`

Sull'area di un `Panel` (e quindi anche di un'applet perché la classe `Applet` è una derivata della classe `Panel`, cioè si potrebbe dire che in fondo è un `Panel` speciale#) o di un `Frame` contenitore di un'applicazione, possono essere posti vari componenti (etichette, campi o aree di input, bottoni, o anche altri `Panel` o `Canvas`, etc.).

Per inserire un componente (un qualunque oggetto derivato da `Component`) in un contenitore (oggetto derivato da `Container`) si ha a disposizione il metodo `add(Component)` della classe `Container`: se il componente non richiede un particolare trattamento ma può essere usato nella sua versione originale con le proprietà che il suo costruttore permette di fissare e con le sole caratteristiche di risposta agli eventi previste nella superclasse, lo si può introdurre direttamente istanziandolo dalla relativa classe dell'AWT, altrimenti bisogna sottoclassarlo per conferirgli le caratteristiche volute.

E' certamente possibile posizionare un componente all'interno del suo contenitore fissando la sua posizione e le sue dimensioni assolute definite in numero preciso di pixel sullo schermo con il metodo `reshape(x, y, w, h)`, ma questo modo, che pur permette una scelta libera e precisa, potrebbe andare bene in una piattaforma determinata (per esempio quella stessa su cui si è sviluppato il programma) e non su altre, dove potrebbe apparire disordinata. Per superare l'inconveniente di poter avere delle risposte diverse sulle varie piattaforme su cui un'applet o un'applicazione deve essere eseguita riguardo a finestre, componenti o font, provvidenzialmente l'AWT dispone di un sistema ad alto livello (nel senso che libera dai dettagli) e flessibile per posizionare i componenti automaticamente secondo determinate disposizioni (*layout*), affidando all'implementazione dell'AWT su ciascuna piattaforma l'adattamento alle caratteristiche specifiche di essa: questa è la ragione dell'esistenza dei manager di layout.

Per disporre gli elementi è utile definire un *layout*, cioè un tipo di disposizione, mediante un **gestore di *layout*** (layout manager) che Java fornisce e che si occupa di sistemare i vari componenti entro un contenitore. La posizione che ogni componente viene ad assumere sul suo contenitore dipende così dal `LayoutManager` definito per quel contenitore (il `LayoutManager` di default è il `FlowLayout` per un `Panel` e il `BorderLayout` per un `Frame`, vedi paragrafi seguenti).sintetizzare#

La posizione e le dimensioni che i componenti vengono ad assumere entro il loro contenitore dipendono dal tipo di layout definito per il contenitore, dall'ordine in cui sono stati aggiunti al contenitore e dalle loro caratteristiche geometriche, che dipendono dalla lunghezza delle stringhe di testo che vi devono apparire, dalla grandezza dei caratteri del font del componente e dalle dimensioni eventualmente prefissate, per esempio per i campi di input.

Per assegnare un determinato gestore di layout ad un contenitore si usa il metodo `setLayout()`.

1.3.12 Disposizione tipo **FlowLayout**

Con il **FlowLayout**, che è quello adottato per default dall'AWT per i contenitori di tipo `Panel`, i componenti sono disposti tutti in fila uno dopo l'altro, nell'ordine da sinistra a destra con cui sono via via aggiunti al contenitore, su righe orizzontali, passando ad una riga successiva quando su una riga non ce ne stanno più.

I componenti sono per default centrati orizzontalmente sulla riga rispetto ai bordi destro e sinistro del contenitore, ma questo allineamento può anche essere stabilito diversamente mediante un argomento del costruttore che può essere dato come `FlowLayout.CENTER`, `FlowLayout.LEFT` o `FlowLayout.RIGHT`, essendo questi valori definiti come variabili di classe nella classe `FlowLayout`.

La distanza di default tra i componenti è di 5 pixel l'uno dall'altro, ma questa distanza, sia orizzontale sia verticale, può essere fissata ad un valore diverso, con argomenti passati ad uno dei costruttori, essendoci due costruttori disponibili, uno senza ed uno con la distanza come argomento.

Costruttori:

FlowLayout()	crea un `FlowLayout` con un allineamento centrato ed una distanza tra i componenti di 5 pixel orizzontalmente (dx) e verticalmente (dy)
FlowLayout(int)	crea un `FlowLayout` con un allineamento centrato e dx=dy=5
FlowLayout(int,int dx,int dy)	crea un `FlowLayout` con allineamento centrato e dx e dy dati

L'istruzione con cui viene fissato per un contenitore un certo layout è del tipo:

```
setLayout(new FlowLayout(FlowLayout.CENTER, 0, 0));
```

Viene fatto qui un esempio di codice che piazza una serie di bottoni su più righe.

```
// H12flowlay.java (F.Spagna) Esempio di disposizione tipo FlowLayout

import java.awt.*;

public class H12flowLay extends java.applet.Applet {

   public void init() {
      setBackground(Color.white);
      setLayout(new FlowLayout());    // puo' essere omessa perche' di default
      for (int n = 0; n < 9; n++)
         add(new Button("Bottone" + n));
   }
}
```

Il risultato, per un'applet cosiffatta, richiamata sull'HTML con:

```
<applet code=botFlowLay.class width=300 height=150></applet>
```

è presentato nella figura 8.12.

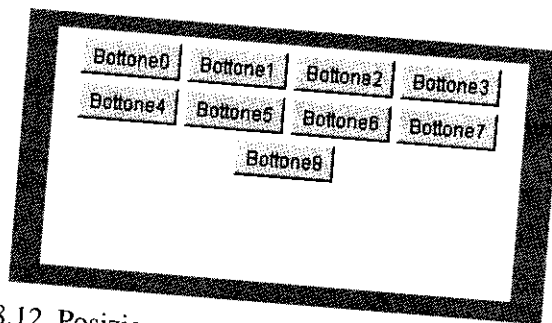



Figura 8.12 Posizionamento di bottoni con il `FlowLayout`.

1.3.13 Disposizione tipo `BorderLayout`

Con il **`BorderLayout`** i componenti sono disposti in relazione ai quattro lati (*border*) del contenitore. Facendo in particolare l'ipotesi che i componenti da sistemare siano dei bottoni, se essi non sono specificamente dimensionati vanno ad occupare tutto il lato al quale sono stati assegnati, con precedenza per quelli orizzontali, che se sono soli nella riga si estendono per tutta la lunghezza del loro lato in priorità rispetto a quelli verticali. I bottoni sui lati hanno uno spessore determinato dallo spazio minimo richiesto dalla scritta su di essi (altezza dei caratteri in un senso e lunghezza della stringa nell'altro), mentre quello al centro occupa tutto lo spazio che gli resta libero una volta che i bottoni lungo i bordi siano stati sistemati (fino a tutta l'area del contenitore se c'è solo un bottone di centro).

Costruttori:

`BorderLayout()`	crea un `BorderLayout` con distanza minima tra i componenti
`BorderLayout(int dx,int dy)`	crea un `BorderLayout` con distanze dx e dy date

Nel programma, dopo aver stabilito:

```
setLayout(new BorderLayout());
```

ogni nuovo bottone viene aggiunto con un metodo di tipo:

```
add("North", new Button("OK"));
```

Il metodo `add()` da adoperare in tal caso è quello nella sua versione che prende un primo argomento costituito da una stringa che si riferisce alla posizione e che può assumere i valori

"North", "South", "West" o "East" a seconda del lato su cui si vuole piazzare il componente (rispettivamente alto, basso, sinistro e destro).

Per imporre una distanza orizzontale e verticale tra i componenti si può usare un costruttore di BorderLayout che prevede le distanze come argomenti, come:

```
setLayout(new BorderLayout(20, 20));
```

così come, se non si vuole che i componenti siano troppo vicini ai bordi e li si vogliono distanziare da essi, si può ricorrere al metodo insets() del componente (vedi paragrafo 7.3.17).

Esempio di codice che piazza un componente (un bottone) per ognuno dei quattro lati ed un quinto al centro:

```
// H13bordlay.java (F.Spagna) Esempio di disposizione tipo BorderLayout
import java.awt.*;

public class H13bordLay extends java.applet.Applet {

   public void init() {
      setBackground(Color.white);
      setLayout(new BorderLayout());
      add("West",   new Button("Bottone W"));
      add("East",   new Button("Bottone E"));
      add("North",  new Button("Bottone N"));
      add("South",  new Button("Bottone S"));
      add("Center", new Button("Bottone C"));
   }
}
```

Il risultato, per l'applet precedente richiamata sull'HTML con:

```
<applet code=botBordLay.class width=300 height=150></applet>
```

è presentato in figura 8.13. Dalla figura risultano chiari i criteri accennati relativamente a lunghezza e spessore dei bottoni. In particolare si vede come i bottoni N e S, essendo su lati orizzontali, vengono ad occupare tutto il loro lato, mentre i bottoni W ed E sui lati verticali devono accontentarsi della lunghezza libera rimasta su quei lati, ed infine come il bottone C viene ad occupare tutto lo spazio restante al centro. Si può osservare anche come gli spessori dei bottoni sui lati verticali si adeguano automaticamente alle dimensioni delle scritte orizzontali che vi devono essere apposte.

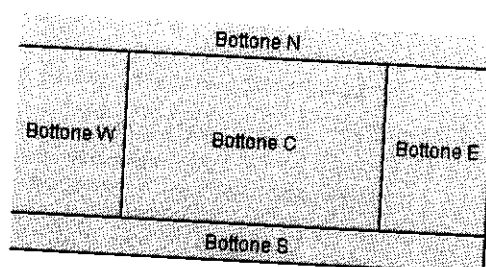



Figura 8.13 Posizionamento di bottoni con il `BorderLayout`.

Altro esempio con Frame:#

1.3.14 Disposizione tipo **GridLayout**

Il **GridLayout** permette una disposizione dei componenti per righe e colonne nelle celle di una griglia a maglie regolari (a questo tipo bisogna ricorrere se si vuole avere un allineamento verticale di componenti su colonne, che non è possibile con il `FlowLayout`). L'ordine con cui i componenti sono piazzati è quello stesso del `FlowLayout`, cioè partendo dalla prima riga in alto e procedendo da sinistra a destra secondo l'ordine con cui vengono via via aggiunti con il metodo `add()`.

Costruttori:#

GridLayout()	crea un `GridLayout`
GridLayout(int r,int c)	crea un `GridLayout` con numero dato di righe e colonne
GridLayout(int r,int c,int dx,int dy)	crea un `GridLayout` con numero di righe e colonne dati e distanze dx e dy tra i componenti assegnate

Per fissare il layout:

```
setLayout(new GridLayout(numRig, numCol, distOriz, distVert));
```

Con un'appropriata suddivisione di un contenitore in una griglia di sottocontenitori a loro volta suddivisi in griglie a maglie più fini si può arrivare ad ottenere un piazzamento abbastanza preciso di ogni componente, anche senza ricorrere ad un posizionamento assoluto assegnato in pixel, che generalmente per le ragioni già dette è sconsigliabile.

Esempio di codice che dispone su 4 colonne una serie di bottoni (oggetti `Button`) e di etichette (oggetti `Label`) alternati:

```
// H14gridlay.java (F.Spagna) Esempio di disposizione tipo GridLayout
import java.awt.*;

public class H14gridLay extends java.applet.Applet {

   public void init() {
      setBackground(Color.white);
      setLayout(new GridLayout(0, 4, 10, 12));
      for (int n = 0; n < 9; n++) {
         add(new Button("Bottone " + n));
         add(new Label("Etichetta " + n));
```

```
        }
    }
}
```

Il risultato, per un'applet così definita e richiamata sull'HTML con:

```
<applet code=botBordLay.class width=300 height=150></applet>
```

è presentato in figura 8.14.

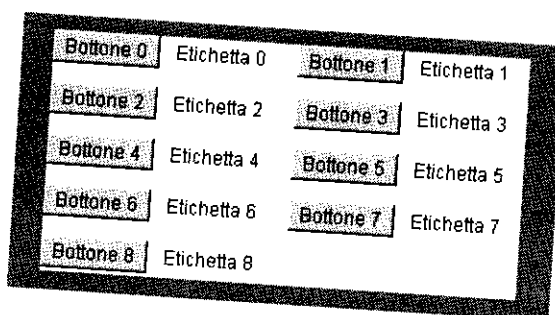



Figura 8.14 Posizionamento di vari componenti su una griglia con il `GridLayout`.

1.3.15 Disposizione tipo `GridBagLayout`

Il gestore di layout rappresentato dalla classe **`GridBagLayout`** permette un allineamento dei componenti verticalmente e orizzontalmente secondo una griglia a maglie regolari come il `GridLayout` ma con la possibilità in più che un componente può occupare più di una maglia, adattandosi ad una serie di suggerimenti dati mediante la classe ausiliaria **`GridBagConstraints`**.

Ogni componente gestito da questo gestore è associato con una sua propria istanza della classe `GridBagConstraints` che definisce come ciascun componente è tracciato all'interno della sua *display area*.

Le caratteristiche volute per ogni componente che si aggiunge alla griglia sono stabilite dal metodo **`setConstraints()`** di `GridBagLayout` che riceve come argomenti la referenza del componente stesso e un oggetto (un'istanza di `GridBagConstraints`) nel quale sono raccolte tutte le regole attraverso il fissaggio di tutta una serie di variabili di istanza di esso. Tra queste variabili ricordiamo:

`gridx e gridy`

che riguardano la posizione orizzontale e verticale nella griglia

`gridwidth e gridheight`

che riguardano le dimensioni orizzontale e verticale in numero di maglie di griglia

fill

che stabilisce se il componente occupa tutto lo spazio orizzontalmente o verticalmente della sua display area (si fissa con una delle variabili di classe GridBagConstraints.NONE, HORIZONTAL, VERTICAL, BOTH)

weightx e weighty

stabiliscono la distribuzione tra i vari componenti degli spazi extra in senso orizzontale e in senso verticale quando si ridimensiona il contenitore (si assegna loro un valore compreso tra 0. e 1.0, mentre il valore di default è 0.)

anchor

che specifica dove viene posto il componente nella sua display area (si fissa con le variabili di classe GridBagConstraints.NORTHWEST, NORTH, etc.)

insets

con esso si stabilisce lo spazio tra il componente e i bordi della sua display area

ipadx e ipady

stabiliscono le aggiunte di spazi interni al componente nelle due direzioni.

Costruttore:

GridBagLayout()	crea un `GridBagLayout`

Metodi:

setConstraints(Component, GridBagConstraints)	stabilisce le `Constraints ?#`

Esempio:

```
// H15gridbag.java (F.Spagna) Esempio di GridBagLayout
import java.awt.*;
public class H15gridbag extends java.applet.Applet {
   public void init() {
      GridBagLayout gb = new GridBagLayout();
      setLayout(gb);
      GridBagConstraints c = new GridBagConstraints();
```

```
        for (int n = 0; n < 3; n++)
            for (int i = 0; i < 3; i++) {
                c.gridx = i;        c.gridy = n;
                c.gridwidth = 1; c.gridheight = 1;
                Button b = new Button("bottone" + n + i);
                gb.setConstraints(b, c);
                add(b);
            }
        c.gridx = 0;        c.gridy = 3;              // 1a colonna e 4a riga
        c.gridwidth = 2; c.gridheight = 1; // larghezza di due colonne
        c.fill = GridBagConstraints.HORIZONTAL;
        Button b1 = new Button("germania");
        gb.setConstraints(b1, c);
        add(b1);

        c.gridx = 3;        c.gridy = 0;              // 4a colonna e 1a riga
        c.gridwidth = 1; c.gridheight = 2;         // altezza di due righe
        c.fill = GridBagConstraints.VERTICAL;
        Button b2 = new Button("bretagna");
        gb.setConstraints(b2, c);
        add(b2);
    }
}
```

In figura 8.15 è mostrato il risultato.

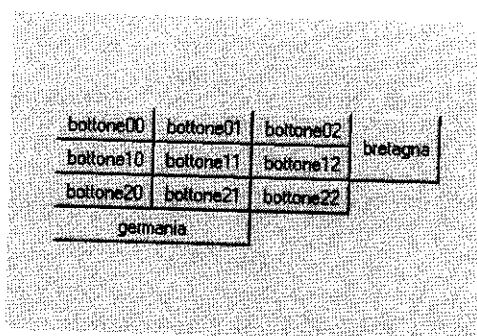



Figura 8.15 Esempio di disposizione di componenti tipo GridBagLayout.

1.3.16 Disposizione tipo `CardLayout`

Il gestore di layout tipo **`CardLayout`** permette di creare per un contenitore diversi altri contenitori (o ***card***) ciascuno con un suo proprio nome, che possono apparire solo uno alla volta sul contenitore principale corrispondentemente al metodo `show(this, Panel)` della classe xx# che di volta in volta può essere chiamato. Ognuno dei contenitori componenti può avere i suoi propri componenti ed un suo proprio layout, così che l'aspetto che il contenitore principale viene ad assumere può essere di volta in volta diverso.

Costruttori:

CardLayout()	crea un `CardLayout` senza distanza tra i componenti
CardLayout(int dx,int dy)	crea un `CardLayout` con distanze dx e dy date

Esempio:#

Figura 8.16 Esempio di disposizione tipo `CardLayout`.

v. libro p.254#

#rivedi seguente

add("primo", pan1);

show(this, "primo");

Usando il metodo add() del contenitore nella forma:

add(String nome, Component comp);

che assegna un nome al componente e poi il metodo show()

1.3.17 Distanza dei componenti dai bordi (`Insets`)

La distanza dei componenti inseriti in un contenitore da ciascuno dei quattro bordi di esso può essere imposta ridefinendo per il contenitore il metodo **`insets()`** che ogni contenitore eredita dalla classe `Container` così:

```
public Insets insets() {
    return new Insets(a, b, c, d);
}
```

ove i quattro argomenti del costruttore di **`Insets`** rappresentano le distanze nell'ordine dal bordo superiore, inferiore, sinistro e destro del contenitore.

Si fa qui di seguito un esempio:#

1.3.17.1 Suddivisione di un contenitore in altri contenitori

L'area di un contenitore, ad esempio un `Panel`, può essere suddivisa in diversi altri contenitori (sottocontenitori), ad esempio altri `Panel` (pannelli annidati o *nested Panels*), per poter così avere maggiore flessibilità nel distribuire i componenti all'interno di esso, potendo ciascuno dei sottocontenitori avere un suo proprio gestore di layout con disposizione particolare dei componenti, e colori delle scritte e del fondo e font particolari. Ne vedremo un esempio nel paragrafo seguente.

Innestando dei sottocontenitori in un contenitore principale e in essi eventualmente altri sottocontenitori e così di seguito, si viene a formare una gerarchia di contenitori e componenti contenuti che, partendo dal contenitore principale, scende via via fino ai componenti più interni. La posizione in questa gerarchia dei componenti determina l'ordine con cui essi vengono disposti sullo schermo in seguito ad un evento di paint e soprattutto il loro comportamento in relazione agli eventi.

Per suddividere un contenitore in altri contenitori si creano prima questi contenitori interni come si farebbe per qualsiasi altro componente e li si aggiungono quindi al contenitore principale con il solito metodo `add()`. Ad esempio per un `Panel`:

```
Panel p1 = new Panel();
Panel p2 = new Panel();
```

vedi esempio di pagina 258 del libro #

Quando un contenitore, ad esempio un `Panel`, contiene un altro contenitore, ad esempio un altro `Panel` (che potremmo chiamare sottopannello o subpanel), se quest'ultimo è definito nel programma con una classe a sé stante si deve evidentemente istanziare, come già detto, nel `Panel` principale un'istanza del subpanel che è un suo costituente, ma se si vuole che il sottopannello possa agire sul pannello per suoi eventi si dovrà creare dentro la classe del subpanel una referenza al pannello principale quando la si definisce perchè il pannello interno possa per suoi eventi comunicare con il suo contenitore invocandone i metodi. Infatti in tal caso bisogna che il subpanel possa conoscere il suo pannello contenitore perché possa passargli … e per questo si deve prevedere che il suo costruttore riceva come argomento la referenza al contenitore. Così, quando si istanzia il subpanel nella classe del contenitore principale, gli si passa come argomento nel costruttore il `this` e lato subpanel la referenza dell'oggetto contenitore come variabile.(semplificare#)

#ESEMPI di libro

1.3.18 Disposizione con vari layout composti

Poichè ad un contenitore si può associare un gestore di disposizione componenti ed un contenitore può comprendere a sua volta diversi altri contenitori, ciascuno con il suo proprio gestore, vari gestori possono essere combinati insieme in cascata e si possono così sfruttare

convenientemente i vantaggi di diversi tipi di *layout* contemporaneamente. Così per sistemare una serie di componenti sull'area di un `Panel` si possono usare diversi altri `Panel`, ciascuno dei quali contiene alcuni dei componenti ordinati secondo un suo particolare *layout*.

Si può fare sullo stesso contenitore (ad esempio un `Panel` o un'`Applet`) una composizione di vari *layout* diversi definendone ciascuno su un suo specifico `Panel` e componendo quindi i vari `Panel` in un *layout* generale.

Esempio di codice che sistema due `Panel` con *layout* di tipo diverso (uno di tipo `Flowlayout` ed uno di tipo `Borderlayout`) sullo stesso contenitore.

```
// H16piuLay.java (F.Spagna) Esempio di disposizione con vari layout
import java.awt.*;

public class H16piuLay extends java.applet.Applet {

   public void init() {

      setBackground(Color.white);

      Panel pf = new Panel();
      pf.setLayout(new FlowLayout());
      for (int n = 0; n < 4; n++)
         pf.add(new Button("Bottone " + n));

      Panel pb = new Panel();
      pb.setLayout(new BorderLayout());
      pb.add("West",   new Button("Bottone W"));
      pb.add("East",   new Button("Bottone E"));
      pb.add("North",  new Button("Bottone N"));
      pb.add("South",  new Button("Bottone S"));
      pb.add("Center", new Button("Bottone C"));

      add("North", pf);
      add("South", pb);
   }
}
```

Il risultato, per un'applet così definita e richiamata sull'HTML con un'applet di dimensioni 300 x 150:

```
<applet code=botBordLay.class width=300 height=150></applet>
```

è presentato in figura 8.17.

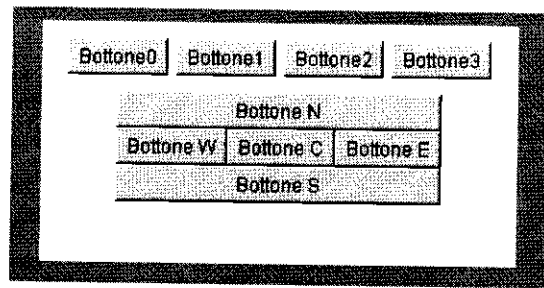



Figura 8.17 Posizionamento di bottoni con due layout composti.

1.4 Eventi

1.4.1 Programmazione ad eventi

Il linguaggio Java è predisposto per una *programmazione ad eventi*. Contrariamente alla **programmazione procedurale**, secondo la quale il programma ha un flusso cosiddetto *top to bottom*, cioè che fa un percorso dalla cima al fondo, nella **programmazione ad eventi** ogni programma (un programma ad eventi è detto in inglese *event driven*) resta continuamente in un *loop* di attesa di eventi esterni. Un evento può essere generato da un'azione dell'utente, come la battuta di un tasto, un clic o lo spostamento del cursore del mouse, la chiusura o l'apertura di una finestra o il suo scorrimento, ma anche da fatti esterni come ad esempio dei dati in arrivo dalla rete o su una porta, etc. Gli eventi via via che si producono vengono posti in una **coda di eventi** dal sistema operativo, che li raccoglie e li smista ai vari programmi, secondo la loro destinazione. Ad ogni ciclo del loop un programma cerca nella propria coda il prossimo evento e, se ne trova uno, attiva il corrispondente metodo previsto in risposta all'evento. Perciò nella classe di definizione dell'applet, o di un'applicazione Java indipendente, possono essere definiti una serie di metodi che devono rispondere ai possibili eventi (tali metodi sono detti in inglese ***event handler***, cioè "che trattano gli eventi"). Nel caso delle applet non è necessario programmare il loop di attesa di eventi, in quanto è il browser stesso che ne fornisce uno.

Gli eventi in Java sono trattati dall'AWT (Abstract Windowing Toolkit) rappresentato dal package `java.awt`, che gestisce i componenti dell'interfaccia grafica utente, ed è in questo package che si trova la classe `Event` che, come vedremo, serve per rappresentare gli eventi.

1.4.2 Modello di gestione degli eventi del JDK 1.0.2

1.4.2.1 Gestione degli eventi e metodi di risposta ad essi

Alcuni degli eventi che possono prodursi durante l'esecuzione di un programma sono trattati automaticamente da Java o dal browser, come ad esempio la chiamata del metodo `paint()` quando un componente deve essere ridisegnato (evento di paint), ma altri eventi devono essere trattati espressamente dal programmatore, come quelli che sono prodotti da un input dell'utilizzatore su un componente UI di interazione, che prevedono sempre delle azioni specifiche conseguenti.

Nella versione 1.0.2 del JDK per la gestione degli eventi che interessano un componente si dispone di vari metodi tra cui i due **`handleEvent()`** e **`action()`** ereditati da ogni componente dalla superclasse `Component`. Quando un evento si produce nel componente il sistema chiama il metodo `handleEvent(Event)` passando ad esso come argomento un'istanza della classe `Event`, generata dal sistema al momento dell'evento, che contiene le informazioni che caratterizzano l'oggetto su cui l'evento è stato prodotto, il tipo di evento, il tempo e la posizione in cui esso si è prodotto.

La classe **Event**, che descrive un evento, infatti possiede, tra le altre, una variabile d'istanza **id** che caratterizza (ne porta in sé l'informazione) il tipo di evento di cui essa rappresenta l'identificatore, secondo una tabella definita da tutta una serie di variabili `final static` di classe definite all'interno della classe stessa `Event`, con le quali può essere confrontata quando si passa un oggetto di tipo `Event` ad un metodo che si occupa del trattamento degli eventi.

Il metodo `handleEvent()` della classe `Component`, superclasse di ogni componente (classe di base all'origine di tutti gli oggetti grafici, che è automaticamente chiamato dal sistema in seguito al verificarsi di un evento, nella sua versione di default, che è quella definita nella superclasse `Component`, possiede già una certa funzionalità di base per la gestione di vari eventi (come ad esempio la chiamata dei metodi relativi a mouse e tastiera). Questo metodo è però in generale ridefinito in un programma:

- per trattare eventi di sistema non compresi nella versione originale del metodo della superclasse `Component`,
- per cambiare il comportamento previsto nel metodo originario,
- per prevedere nuovi tipi di eventi.

Il metodo `handleEvent()`, tra i vari metodi per il trattamento degli eventi, è quello più generale che permette di intercettare e trattare gli eventi di ogni tipo sui componenti e viene usato generalmente nella forma seguente, in cui viene fatto un test, con uno `switch` (oppure una serie di `if`), sul tipo di evento mediante l'identificatore dell'evento, e più precisamente la variabile di istanza `id` dell'oggetto di tipo `Event`, contenente le caratteristiche dell'evento, che il sistema operativo crea e passa come argomento al metodo per ogni evento che si produce. Nella classe `Event` sono definite una serie di variabili (in effetti costanti `final`) di classe (`static`), che, essendo intere, si prestano ad essere usate in uno `switch` (vedi tabella seguente).

Identificatori di eventi di mouse e tastiera:

Event.MOUSE_DOWN	evento generato quando si preme il bottone del mouse sul componente
Event.MOUSE_UP	evento generato quando si rilascia il bottone del mouse sul componente
Event.MOUSE_MOVE	evento generato quando si muove il cursore del mouse sul componente
Event.MOUSE_DRAG	evento generato quando si muove il cursore del mouse sul componente con il bottone premuto
Event.MOUSE_ENTER	evento generato quando il cursore del mouse entra nel componente
Event.MOUSE_EXIT	evento generato quando il cursore del mouse esce dal componente
Event.KEYPRESS	evento generato quando si preme un tasto
Event.KEYRELEASE	evento generato quando si rilascia un tasto
Event.KEYACTION	evento generato quando è effettuata un'azione su un tasto (un'azione consiste in una pressione seguita da un rilascio)

Il metodo `handleEvent()` non fa altro che chiamare a seconda dell'evento il metodo specifico dell'evento stesso, come si può vedere bene dalla sua definizione tratta dal JDK e qui sotto riportata integralmente:

```
public boolean handleEvent(Event evt) {

   switch (evt.id) {

     case Event.MOUSE_ENTER:
       return mouseEnter(evt, evt.x, evt.y);

     case Event.MOUSE_EXIT:
       return mouseExit(evt, evt.x, evt.y);

     case Event.MOUSE_MOVE:
       return mouseMove(evt, evt.x, evt.y);

     case Event.MOUSE_DOWN:
       return mouseDown(evt, evt.x, evt.y);

     case Event.MOUSE_DRAG:
       return mouseDrag(evt, evt.x, evt.y);

     case Event.MOUSE_UP:
       return mouseUp(evt, evt.x, evt.y);

     case Event.KEY_PRESS:
     case Event.KEY_ACTION:
       return keyDown(evt, evt.key);

     case Event.KEY_RELEASE:
     case Event.KEY_ACTION_RELEASE:
       return keyUp(evt, evt.key);

     case Event.ACTION_EVENT:
       return action(evt, evt.arg);
     case Event.GOT_FOCUS:
       return gotFocus(evt, evt.arg);
     case Event.LOST_FOCUS:
       return lostFocus(evt, evt.arg);
   }
   return false;
}
```

Nell'esempio seguente si riporta una ridefinizione del metodo in cui sono considerati solo alcuni tipi di eventi.

```
boolean handleEvent(Event ev) {
   switch (ev.id) {
      case Event.MOUSE_DOWN : /* istruzioni */ ; break;
      case Event.MOUSE_UP :   /* istruzioni */ ; break;
      case Event.MOUSE_MOVE : /* istruzioni */ ; break;
      case Event.ACTION_EVENT :
         if (ev.target == bottone1) /* istruzioni */ ;
         if (ev.target == bottone2) /* istruzioni */ ;
         if (ev.target == bottone3) /* istruzioni */ ;
```

```
            break;
        }
        return super.handleEvent(ev); // recupera la funzionalita' della superclasse
    }
```

#INSERIRE ULTIMI 3 righi pag.234

Nel ridefinire il metodo in una nuova classe si viene a perdere la funzionalità originale contenuta nel metodo della superclasse, e se si vuole recuperarla, basta richiamare il metodo della superclasse dopo aver definito specificamente il trattamento di certi eventi nella sottoclasse: è quello che viene fatto nell'esempio che segue in questo paragrafo.

L'ultima istruzione del metodo nell'esempio precedente, nella quale viene invocato il metodo `handleEvent()` della superclasse, è, nella maggior parte dei casi, necessaria in quanto tutti gli altri eventi previsti nella superclasse (per esempio il `MouseDown()`) che non sono trattati esplicitamente nel metodo che si sta scrivendo potranno essere poi trattati nella loro versione di default della superclasse `Component`, altrimenti il loro trattamento andrebbe perso.

Per trattare gli eventi di tipo mouse e tastiera si possono usare direttamente i metodi specifici tipo `mouseDown()` e `keyDown()` che riportiamo nei paragrafi seguenti e che non richiedono un controllo preliminare sul tipo di evento in quanto sono già specifici di eventi determinati. Per eventi di tipo action (quelli dei componenti di tipo UI) si può invece usare direttamente il metodo `action()`.

1.4.2.2 Metodi specifici rispondenti agli eventi del mouse

Eventi del mouse sono prodotti da una pressione (clic) sul tasto (quello di sinistra se il mouse ne ha più di uno) o spostamenti del cursore del mouse nell'area del componente, o anche azioni combinate di clic e spostamenti. La classe `Component` contiene diversi metodi di risposta agli eventi prodotti da azioni sul mouse da parte dell'utente ai quali sono passati, oltre al solito oggetto di tipo `Event`, altri due argomenti che sono le coordinate `x` e `y` (in pixel) della posizione del cursore del mouse al momento dell'evento (`x` da sinistra ed `y` dall'alto), che comunque sono anche contenuti nell'oggetto di tipo `Event`.

Vedi esempio#

Quando si parla di tasto del mouse ci si riferisce, nel caso di PC al tasto sinistro e nel caso di Macintosh al solo tasto presente sul mouse. Il tasto destro non è considerato in Java per ragioni di generalità interpiattaforma (il Mac non l'ha).

Presentiamo qui di seguito i metodi specifici per gli eventi del mouse.

Il metodo **`mouseDown(Event ev, int x, int y)`** è chiamato ogni volta che il bottone sinistro del mouse viene premuto, con il cursore posizionato dentro l'area del componente. Come viene detto a proposito del metodo seguente, non è con questo evento che generalmente si fa partire nel programma l'azione conseguente alla pressione di un bottone o ad una scelta di menù; con esso invece si fa partire un eventuale comando continuo che deve durare fintantochè il tasto resta premuto (per esempio tipicamente la crescita progressiva col tempo di un valore).

Il metodo **mouseUp(Event ev, int x, int y)** è chiamato ogni volta che il bottone sinistro del mouse viene rilasciato, con il cursore posizionato dentro l'area del componente. Generalmente è con questo evento che si rende operante nel programma l'azione conseguente alla pressione di un bottone o ad una scelta di menù, piuttosto che con `mouseDown()`, perché così si permette all'utente di cambiare idea ancora quando ha già premuto il tasto uscendo con il cursore fuori del bottone prima di rilasciare il tasto del mouse.

Il metodo **mouseMove(Event ev, int x, int y)** è chiamato ogni volta che, con il bottone sinistro del mouse sollevato, si muove il cursore dentro l'area del componente spostandosi di almeno un pixel in qualsiasi direzione (un evento è generato per ogni pixel di spostamento, per cui uno spostamento del cursore di una certa ampiezza scatena una sequenza di eventi di questo tipo).

Il metodo **mouseDrag(Event ev, int x, int y)** è chiamato ogni volta che, con il bottone sinistro del mouse premuto, si muove il cursore dentro l'area del componente spostandosi di almeno un pixel in qualsiasi direzione (anche in questo caso un evento è generato per ogni pixel di spostamento, per cui anche in questo caso uno spostamento del cursore di una certa ampiezza scatena una sequenza di eventi di questo tipo).

Il metodo **mouseEnter(Event ev, int x, int y)** è chiamato ogni volta che il cursore entra nell'area del componente, con `x` e `y` che rappresentano le coordinate del punto di entrata.

Il metodo **mouseExit(Event ev, int x, int y)** è chiamato ogni volta che il cursore esce dall'area del componente, con `x` e `y` che rappresentano le coordinate del punto di uscita.

1.4.2.3 Metodi specifici di risposta agli eventi di tastiera

La battuta di un tasto sulla tastiera da parte dell'utilizzatore costituisce un evento che può essere intercettato con il metodo **keyDown(Event ev, int tasto)**, che è chiamato ogni volta che viene premuto un tasto mentre l'applet è attiva (questa precisazione è importante perché la battuta del tasto non viene captata se l'applet non ha il *focus* e questo si può ottenere con un clic del mouse su di essa). Il sistema passa come argomento a tale metodo il valore ASCII (un intero) del carattere relativo al tasto e il carattere corrispondente al valore ASCII può poi essere ottenuto con un cast di tipo `(char)tasto`.

Esempio:#

Oltre ai caratteri alfanumerici scrivibili, il tasto può però riferirsi anche ad altri caratteri speciali, come il Return, il Tab, le frecce nei vari sensi, etc. Nella classe `Event` sono definite tutta una serie di variabili di classe `static` (in effetti delle costanti `final`) che rappresentano i valori di tasti speciali (quelli non scrivibili), come:

Event.HOME, Event.END, Event.UP, Event.DOWN, Event.LEFT, Event.RIGHT.

Si noti che, essendo i valori dei tasti degli interi, si prestano ad essere usati in uno `switch` quando si vuole andare a vedere che tasto è stato battuto.

Esempio:#

1.4.2.4 Metodi per i tasti Shift, Ctrl e Alt

I metodi **`shiftDown()`**, **`controlDown()`** e **`metaDown()`** possono essere usati per verificare se i tasti rispettivamente di maiuscole (Shift), Control e Alt (Meta in Unix) sono premuti nello stesso momento in cui si sta intercettando un certo evento, per esempio su un altro tasto (combinazione di più tasti) o su un'operazione del mouse.

Esempio:#

1.4.2.5 Metodi relativi al *focus* su un componente

Il metodo **`getFocus()`** è chiamato ogni volta che il

Il metodo **`gotFocus()`** è chiamato ogni volta che il

Il metodo **`lostFocus()`** è chiamato ogni volta che il

Il metodo **`hasFocus()`** restituisce `true` se il componente ha il focus di tastiera

1.4.2.6 Eventi dei componenti UI di interazione utente e metodo `action()`

I componenti di interazione con l'utente (componenti UI) producono degli eventi chiamati **azioni**. La variabile di istanza `id` dell'oggetto di tipo `Event` creato e passato ai metodi di trattamento degli eventi dal sistema, ha in tal caso il valore di `Event.ACTION_EVENT`. Eventi di questo tipo possono essere intercettati, oltre che con il metodo generale `handleEvent(Event)`, anche più specificamente con il metodo **`action()`** del contenitore del componente interessato all'azione:

```
public boolean action(Event ev, Object arg)
```

che riceve un primo argomento di tipo `Event`, che, come per i metodi relativi agli eventi di mouse e tastiera, gli viene passato dal sistema e che caratterizza l'evento, ed un secondo argomento che è un oggetto tipico di ciascun tipo di componente UI che contiene le informazioni specifiche ritenute necessarie per esso. Quest'oggetto può quindi essere di varia natura (cioè di classe diversa) a seconda del componente che lo invia, e in particolare:

- i bottoni (oggetti `Button`) producono un'azione se sono premuti e inviano come secondo argomento al metodo un'informazione sotto forma di una stringa che rappresenta l'etichetta del bottone e può servire al suo riconoscimento,

- le `Checkbox` (dei due tipi) producono un'azione quando sono selezionate e il secondo argomento passato è sempre `true`,

- i menù di tipo `Choice` o `List` producono un'azione quando una voce è selezionata e il secondo argomento è la voce stessa, cioè la scelta fatta, come stringa,

- i campi di input di testo di tipo `TextField` producono un'azione quando l'utente batte il tasto Return mentre il `TextField` ha il focus ed è quindi in stato di edizione, cioè di ricevimento dati,

- i menù producono un'azione quando sono selezionati e il secondo argomento passato è il `MenuItem scelto` sotto forma di stringa.

Poiché il metodo `action()` può ricevere eventi da componenti diversi bisogna fare una verifica sul tipo di oggetto (precisamente la classe) del componente che ha prodotto l'evento e questa informazione è riposta nella variabile di istanza **`target`** dell'oggetto `Event` ricevuto come primo argomento dal metodo. La classe dell'oggetto che ha prodotto l'evento può essere trovata servendosi dell'operatore `instanceof` (vedi paragrafo 2.6) con istruzioni di verifica del tipo:

```
if (ev.target instanceof Button)
```

che precede il blocco di istruzioni relative alle operazioni prodotte dal bottone. Ecco un esempio:

```
public boolean action(Event ev, Object arg) {

   if (ev.target instance of Button) {
      /* istruzioni1 */
      return true;
   }
   if (ev.target instance of TextField) {
      /* istruzioni2 */
      return true;
   }
   if (ev.target instance of Choice) {
      /* istruzioni3 */
      return true;
   }
}
```

#esempio completo#

Esempio: 5 bottoni fanno 5 scritte#

Oppure diversi v.pag.257 del libro#

Un altro modo per individuare la causa dell'evento, indicato per esempio per riconoscere quale di una serie di bottoni ha scatenato l'evento è quello di adoperare sempre la variabile d'istanza `target` di `Event`, di classe `Object` e quindi molto generica, che indica appunto il componente bersaglio (*target*), quello cioè sul quale l'evento è avvenuto, ma verificando la sua referenza con quella dei vari componenti possibili. Ad esempio:

esempio#

Vedremo (nel paragrafo 8.4.3.#) come nel trattamento degli eventi di tipo 1.1 quest'oggetto è sostituito con quello ottenuto dal metodo `AWTEvent.getSource()`.

Eventi relativi alle finestre definiti come variabili di classe nella classe `Event`:

WINDOW_DESTROY	evento prodotto quando una finestra viene chiusa
WINDOW_EXPOSE	evento prodotto quando una finestra coperta viene ripresentata
WINDOW_ICONIFY	evento prodotto quando una finestra è ridotta a icona
WINDOW_DEICONIFY	evento prodotto quando una finestra ridotta a icona è rivisualizzata
WINDOW_MOVED	evento prodotto quando una finestra è spostata sullo schermo

v. pag.266 del libro per variabili di classe per eventi action, list, scroll#

ACTION_EVENT	evento prodotto da un componente UI
KEY_ACTION	evento prodotto da un'azione su un `TextField`
LIST_SELECT	evento prodotto quando una voce di una `List` è selezionata
LIST_DESELECT	evento prodotto quando una voce di una `List` è deselezionata
SCROLL_ABSOLUTE	evento prodotto quando si sposta il cursore di una `ScrollBar`
SCROLL_LINE_DOWN	evento prodotto quando è azionato uno spostamento giù piccolo
SCROLL_LINE_UP	evento prodotto quando è azionato uno spostamento su piccolo
SCROLL_PAGE_DOWN	evento prodotto quando è azionato uno spostamento giù grande
SCROLL_PAGE_UP	evento prodotto quando è azionato uno spostamento su grande

1.4.2.7 Eventi in componenti annidati uno dentro l'altro

In caso di componenti annidati uno dentro l'altro un evento (per esempio un clic del mouse) viene sempre ricevuto dapprima dal componente più interno coinvolto nell'evento, il quale può trattare o no l'evento e lo può poi passare o no al componente più esterno, che lo può a sua volta trattare a modo suo o no e può passarlo al componente ancora più esterno e così via fino al contenitore principale. Quando in questa catena un componente intercetta un evento due sono i casi che si possono verificare: o lo tratta (o no) e tutto finisce là, oppure lo tratta (o no) e lo passa al contenitore di livello superiore: nel primo caso il metodo (*event handler*) che tratta l'evento,

`handleEvent()` in generale o altri metodi specifici come quelli di mouse e tastiera, deve restituire `true` e in questo modo l'evento non è più trasmesso in su nella scala gerarchica, e nel secondo caso deve restituire `false` perchè l'evento sia trasmesso al contenitore di livello superiore. Ecco la ragione per cui i metodi di trattamento degli eventi (cosiddetti *event handler*) devono restituire tutti un valore booleano.

Fai esempio con bottone e cerchio blu sotto: x e y non nel primo metodo ma si nel secondo#

1.4.3 Modello di gestione degli eventi del JDK 1.1

1.4.3.1 Modello di trattamento degli eventi per delega

Il modello di gestione degli eventi ha subìto un'evoluzione passando dalla versione JDK 1.0.2 a quella JDK 1.1, con il quale è adottato un modello "per delega" in cui i componenti utilizzano una classe esterna per la gestione degli eventi. In questo modello gli eventi sono oggetti della classe `EventObject` appartenente al package `java.util`.

1.4.3.2 Interfacce "listener"

Si riportano nei riquadri seguenti le interfacce, contenute nel package `java.awt.event`, "listener" degli eventi particolari, con i loro metodi, il tipo di oggetto rappresentante l'evento che ricevono come argomento e l'indicazione degli eventi in seguito ai quali essi sono invocati.

`ActionListener` (riceve eventi di tipo **action**)

`actionPerformed(ActionEvent)`	se avviene un'action

`AdjustmentListener` (riceve eventi di tipo **adjustment**)

`adjustmentValueChanged(AdjustmentEvent)`	se è cambiato il valore dell'adjustable

`ComponentListener` (riceve eventi di tipo **component**)

`componentHidden(ComponentEvent)`	se un componente è stato nascosto
`componentMoved(ComponentEvent)`	se un componente è stato mosso
`componentResized(ComponentEvent)`	se un componente è stato ridimensionato
`componentShown(ComponentEvent)`	se un componente è stato mostrato

`ContainerListener` (riceve eventi di tipo **container**)

`componentAdded(ContainerEvent)`	se un componente è stato aggiunto al contenitore
`componentRemoved(ContainerEvent)`	se un componente è stato rimosso dal contenitore

`FocusListener` (riceve eventi di tipo "**keyboard focus**" su un componente)

`focusGained(FocusEvent)`	se un componente prende il focus di tastiera
`focusLost(FocusEvent)`	se un componente perde il focus di tastiera

`ItemListener` (riceve eventi di tipo **item**)

`itemStateChanged(ItemEvent)`	se lo **stato di un item** è stato cambiato

`KeyListener` (riceve eventi di **tastiera** (keyboard))

`keyPressed(KeyEvent)`	se un tasto è stato premuto
`keyReleased(KeyEvent)`	se un tasto è stato rilasciato
`keyTyped(KeyEvent)`	se un tasto è stato battuto

`MouseListener` (riceve eventi di **mouse** su un componente)

`mouseClicked(MouseEvent)`	se il mouse è stato clickato su un componente
`mouseEntered(MouseEvent)`	se il cursore del mouse entra in un componente
`mouseExited(MouseEvent)`	se il cursore del mouse esce da un componente
`mousePressed(MouseEvent)`	se un bottone del mouse è stato premuto su un componente
`mouseReleased(MouseEvent)`	se un bottone del mouse è stato rilasciato su un componente

`MouseMotionListener` (riceve eventi di **movimento di mouse** su un componente)

`mouseDragged(MouseEvent)`	se un bottone del mouse è premuto su un componente e quindi "dragged"
`mouseMoved(MouseEvent)`	se il bottone del mouse è stato mosso su un componente (senza premere bottoni)

`TextListener` (riceve eventi tipo **testo**)

`textValueChanged(TextEvent)`	se il valore del testo è cambiato

`WindowListener` (riceve eventi di tipo **window**)

`windowActivated(WindowEvent)`	se una window è attivata
`windowClosed(WindowEvent)`	se una window è stata chiusa
`windowClosing(WindowEvent)`	se una window è in corso di chiusura
`windowDeactivated(WindowEvent)`	se una window è de-attivata
`windowDeiconified(WindowEvent)`	se una window è de-iconificata
`windowIconified(WindowEvent)`	se una window è iconificata

windowOpened(WindowEvent)	se una window è stata aperta

1.4.3.3 Classi di eventi e Adapter

ActionEvent (evento di tipo action)

AdjustmentEvent

ComponentAdapter

ComponentEvent

ContainerAdapter

ContainerEvent

FocusAdapter

FocusEvent

InputEvent

ItemEvent

KeyAdapter

KeyEvent

MouseAdapter

MouseEvent

MouseMotionAdapter

PaintEvent

TextEvent

WindowAdapter

WindowEvent

Esempi

Ma il ruolo di *listener* degli eventi di un componente può essere assunto anche dal contenitore stesso del componente, se lo si implementa come tale alla definizione e si passa il `this` come argomento al metodo che aggiunge il listener: questo è quanto è stato fatto nell'esempio 7.x.y in cui si considera un'applet contenente delle `Chekbox` facenti parte di un `ChekboxGroup`, con il loro listener di tipo `ItemListener`, il cui ruolo è assunto dall'applet stessa, e l'evento `ItemEvent`.

```
// H17CB.java (F.Spagna) Esempio con contenitore stesso come listener di evento

import java.awt.*;
import java.awt.event.*;
```

```
public class H17CB extends java.applet.Applet implements ItemListener {
   Label lab = new Label("zero");
   String s[] = { "uno", "due", "tre" };
   CheckboxGroup cg = new CheckboxGroup();
   Checkbox cb[] = new Checkbox[5];
   public void init() {
      for (int n = 0; n < 3; n++) {
         cb[n] = new Checkbox(s[n], cg, false);
         add(cb[n]);
         cb[n].addItemListener(this);
      }
      lab.setForeground(Color.red);
      add(lab);
   }
   public void itemStateChanged(ItemEvent e) {
      lab.setText(cg.getSelectedCheckbox().getLabel());
   }
}
```

In figura 8. è riportato l'aspetto dell'applet.

```
// H18evbot11.java  F.Spagna  Esempio di gestione eventi in JDK 1.1
// 01-16.12.98 (inizio 16 dic 1998)

import java.awt.*;                                               // per la grafica
import java.awt.event.*;                   // per la gestione degli eventi tipo 1.1

public class H18evbot11 extends Frame {//applicazione costituita da un Frame
   Button bot = new Button("+1");
   Label lab = new Label("0");                                   // con un bottone
   int cont = 0;                                               // con un'etichetta
   H18evbot11() {                           // e avente una variabile contatore
      setLayout(new FlowLayout());                             // il costruttore...
      add(lab);                                            // fissa la disposizione
      add(bot);                                            // aggiunge l'etichetta
      bot.addActionListener(new ascoltBot(this));            // aggiunge il bottone
   }                                                  // e l'ascolto del bottone
   static public void main(String args[]) {             // il metodo di partenza
      evbot11 app = new evbot11(); //crea un'istanza di applicazione (Frame)
      app.pack();              // dimensiona il Frame per contenere gli elementi
      app.show();
   }                                                    // rende visibile il Frame
}
class ascoltBot implements ActionListener  {      // l'ascoltatore di bottone
   H18evbot11 evb;                      // contiese una referenza dell'applicazione
   ascoltBot(H18evbot11 ap) {              // all'istanziazione dell'ascoltatore...
      evb = ap;        // riceve la referenza all'applicazione come argomento
   }
   public void actionPerformed(ActionEvent e) {           // quando evento action
      evb.lab.setText("" + ++evb.cont); // scrive su etich.dell'applicazione
   }
}


// H19evbot11.java  F.Spagna  Esempio di gestione eventi in JDK 1.1
// 01-16.12.98 (inizio 16 dic 1998)
```

```
import java.awt.*;
import java.awt.event.*;                                        // per la grafica
                                   // per la gestione degli eventi tipo 1.1

public class H19evbot11 extends Frame {//applicazione costituita da un Frame
   Button bot = new Button("+1");
   Label lab = new Label("0");                                 // con un bottone
   int cont = 0;                                            // con un'etichetta
   H19evbot11() {                           // e avente una variabile contatore
      setLayout(new FlowLayout());                           // il costruttore...
      add(lab);                                        // fissa la disposizione
      add(bot);                                        // aggiunge l'etichetta
                                                         // aggiunge il bottone
class ascoltBot implements ActionListener  {      // l'ascoltatore di bottone
   H19evbot11 evb;                    // contiese una referenza dell'applicazione
   ascoltBot(H19evbot11 ap) {           // all'istanziazione dell'ascoltatore...
      evb = ap;         // riceve la referenza all'applicazione come argomento
   }
   public void actionPerformed(ActionEvent e) {          // quando evento action
      evb.lab.setText("" + ++evb.cont); // scrive su etich.dell'applicazione
   }
}

      bot.addActionListener(new ascoltBot(this)); // e l'ascolto del bottone


   }
   static public void main(String args[]) {              // il metodo di partenza
      H19evbot11 app = new h19evbot11(); // crea istanza di applicaz.(Frame)
      app.pack();              // dimensiona il Frame per contenere gli elementi
      app.show();
   }                                                 // rende visibile il Frame
}
```

In figura 8. è riportato l'aspetto dell'applet.

La maggior parte delle interfacce listener considerano vari sottotipi di eventi, per ciascuno dei quali è previsto un metodo astratto da implementare nella classe che implementa quell'interfaccia, per esempio `MouseListener` prevede l'implementazione di `mouseClicked()`, `mousePressed()`, `mouseReleased()`, `mouseEntered()` e `mouseExited()`. Se non si è interessati ad implementare tutti i metodi dell'interfaccia, quelli non importanti sono lasciati vuoti, ma questo appesantisce il codice inutilmente: in tal caso per evitare ciò si utilizzano i cosiddetti adapter, che sono classi che si occupano di implementare le interfacce con tutti metodi vuoti. Così, anzichè implementare (con implemets) l'interfaccia nella classe listener, si fa discendere questa (con extend) dal relativo adapter, potendocisi così limitare ad implementare solo i metodi necessari con un *overriding*.

Gli adapter:

```
ComponentAdapter
ContainerAdapter
FocusAdapter
KeyAdapter
MouseAdapter
MouseMotionAdapter
WindowAdapter
```

Esempio:

```
// H20evmouse.java (F.Spagna) Eventi su un bottone con JDK 1.1
// 01-25.04.99 (inizio 25 aprile 1999)

import java.awt.*;
import java.awt.event.*;

public class H20evmouse extends java.applet.Applet implements MouseListener {

   Label lab;

   public void init() {
      addMouseListener(this);            // si mette un ascoltatore a se stessa
      add(lab = new Label("inizio     "));   // etichetta per vedere l'effetto
   }
   public void mouseClicked(MouseEvent e) {
      lab.setText("" + e.getX() + ", " + e.getY());         // se premuto mouse
   }
   public void mousePressed(MouseEvent e) {} //metodi senza effetto ma necess.
  public void mouseReleased(MouseEvent e) {}
   public void mouseEntered(MouseEvent e) {}
   public void mouseExited(MouseEvent e) {}
}
```

1.4.3.4 Eventi del mouse

Esempio di un'applet che evidenzia con delle scritte i vari eventi del mouse via via che vengono rilevati.

```
// H00mouse088.java F.Spagna (inizio 26 agosto 1999)
//
// Operazioni con il mouse
// 01-26.08.99

import java.awt.*;
import java.awt.event.*;

public class H00mouse088 extends java.applet.Applet   // la stessa applet
                     implements MouseListener,      // sente i clic del mouse
                                MouseMotionListener {     // i suoi movimenti
   int tipoEvento;
   int x, y;
   String s[] = {"pressed", "released", "dragged", "clicked",
                 "entered", "exited", "moved" };
   public void init() {
      addMouseListener(this);              // delega l'applet a sentire il mouse
      addMouseMotionListener(this);  // delega applet a sentire il movim.mouse
   }
   public void paint(Graphics g) {
      g.drawString(s[tipoEvento], x, y);        // disegna secondo rettangolo
   }
   void scrivi(MouseEvent me, int tipoEvento) {
```

```
        this.tipoEvento = tipoEvento;
        x = me.getX();                    // coordinate del mouse quando premuto
        y = me.getY();
        repaint();
    }
    public void mousePressed(MouseEvent me) {               // quando mouse premuto
        scrivi(me, 0);
    }
    public void mouseReleased(MouseEvent me) {
        scrivi(me, 1);
    }
    public void mouseDragged(MouseEvent me) {               // quando mouse trascinato
        scrivi(me, 2);
    }
    public void mouseClicked(MouseEvent me) {
        scrivi(me, 3);
    }
    public void mouseEntered(MouseEvent me) {
        scrivi(me, 4);
    }
    public void mouseExited(MouseEvent me)  {
        scrivi(me, 5);
    }
    public void mouseMoved(MouseEvent me)   {
        scrivi(me, 6);
    }
}
```

Nella figura 8.xx si vede l'applet in funzione.

#

Figura 8.xx Applet che rileva gli eventi del mouse.

Un altro esempio interessante può essere il seguente, che permette la selezione di un'area di un'immagine con il mouse.

```
// H00mouseSelez086.java F.Spagna (inizio 14 agosto 1999)
// Selezione di un'area con il mouse
// 01-22.08.99

import java.awt.*;
import java.awt.event.*;

public class H00mouseSelez086 extends java.applet.Applet   // la stessa applet
                        implements MouseListener,       // sente i clic del mouse
                                   MouseMotionListener {      // i suoi movimenti
   Image mappa;                                    // immagine mappa da zoomare
   Image imgMemo;                                        // immagine in memoria
   Graphics gr;                          // contesto grafico immagine in memoria
   int x0, y0;                              // coordinate primo punto selezionato
   int x,  y;                             // coordinate secondo punto selezionato
   int x1, y1;    // coordinate vertice alto a sinistra rettangolo selezionato
```

```
    int b,  h;                              // base e altezza rettangolo selezionato
    boolean primoPunto = true;       // vero se primo punto deve essere selezionato

    public void init() {
       int bas = 300, alt = 200;                    // dimensioni immagine in memoria
       mappa = getImage(getCodeBase(), "Italia.gif");        // carica l'immagine
       imgMemo = createImage(bas, alt);                   // crea immagine in memmoria
       gr = imgMemo.getGraphics();            // contesto garfico immagine in memoria
       gr.drawImage(mappa, 0, 0, this);             // disegna l'immagine in memoria
       addMouseListener(this);                   // delega l'applet a sentire il mouse
       addMouseMotionListener(this);   // delega applet a sentire il movim.mouse
    }
    public void paint(Graphics g) {
          g.drawImage(imgMemo, 0, 0, this);                // disegna l'immagine reale
          x1 = Math.min(x0, x);    y1 = Math.min(y0, y);// vertice alto sinistro
          b  = Math.abs(x - x0);  h = Math.abs(y - y0);     // lati solo positivi
          g.setColor(Color.blue);                                 // rettangolo blu
          g.drawRect(x1, y1, b, h);                  // disegna rettangolo selezionato
          g.drawRect(x1+1, y1+1, b-2, h-2);           // disegna secondo rettangolo
    }
    public void update(Graphics g) {
       paint(g);
    }
    public void mousePressed(MouseEvent me) {               // quando mouse premuto
       if (primoPunto) {               // se il primo punto non ancora selezionato
          x0 = me.getX();                    // coordinate del mouse quando premuto
          y0 = me.getY();
          primoPunto = false;                             // annota la selezione fatta
       }
    }
    public void mouseReleased(MouseEvent me) {
       primoPunto = true;
    }
    public void mouseDragged(MouseEvent me) {           // quando mouse trascinato
       x = me.getX();          // coordinate ad ogni istante del mouse trascinato
       y = me.getY();
       repaint();                    // ridisegna immagine e rettangolo selezionato
    }
    public void mouseClicked(MouseEvent me) { }           // metodi non utilizzati
    public void mouseEntered(MouseEvent me) { }
    public void mouseExited(MouseEvent me)  { }
    public void mouseMoved(MouseEvent me)   { }
}
```

La figura 8.xx mostra l'applet durante una selezione.

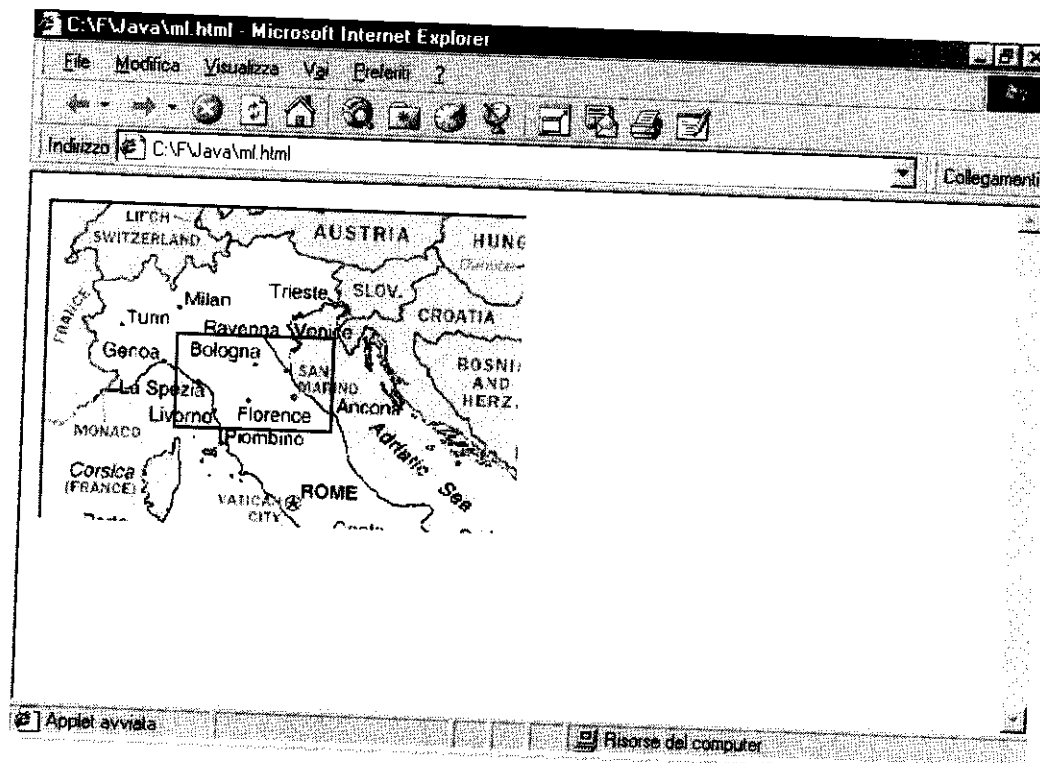



Figura 8.xx Applet con area selezionabile mediante il mouse.

2. Grafica in Java ed eventi

2.1 Grafica dell'AWT

2.1.1 Disegno su un componente senza il metodo `paint()`

Per disegnare qualcosa su un componente tipo `Panel` (e in particolare su un'applet) o su un `Canvas` bisogna avere a disposizione il suo contesto grafico, che è un oggetto di tipo `Graphics` e che può essere considerato come lo strumento di disegno sul componente mediante i suoi metodi grafici. Di solito viene adoperato il metodo `paint(Graphics)` del componente cui il sistema passa il contesto grafico come un oggetto di tipo `Graphics`, ma questo non è strettamente necessario, in quanto basterebbe ottenere il contesto grafico del componente con il metodo **`getGraphics()`** della superclasse `Component`. Per disegnare per esempio un rettangolo le istruzioni sarebbero quindi:

```
Graphics g = getGraphics();
g.drawRectangle();
```

che possono essere sintetizzate in maniera compatta da un'unica istruzione:

```
getGraphics().drawRectangle();
```

nella quale il primo metodo restituisce il contesto grafico di tipo `Graphics` del componente (l'oggetto sul quale si disegna, ad esempio l'applet) con il quale si può disegnare il rettangolo con il metodo `drawRectangle()` proprio della classe `Graphics`.

Esempio:#

Ma senza un metodo `paint()` questo disegno avviene solo una volta e l'applet non è ridisegnata ogni qualvolta una finestra va a ricoprirne il disegno e poi lo scopre o quando da parte dell'utente vengono cambiate le dimensioni della finestra del browser.

2.1.2 Disegno con il metodo `paint()`

E' con il metodo **`paint(Graphics)`** che in un'applicazione in quanto `Frame` o in un'applet in quanto `Panel` (sia `Frame` sia `Panel` ereditano il metodo `paint()` dalla superclasse `Component`) si disegna generalmente qualcosa sullo schermo ogni volta che ciò viene richiesto dalle operazioni che l'utente fa sulle finestre. Questo metodo riceve come argomento un oggetto di tipo `Graphics` (contesto grafico) che gli viene passato dall'interprete, che contiene tutte le informazioni grafiche (colore, font, etc.) richieste dal disegno e che possiede i metodi che servono per disegnare.

La classe **Graphics** del package `java.awt` dispone di vari metodi (primitive grafiche) per la scrittura di stringhe, il disegno di linee e figure geometriche semplici o la riproduzione di immagini bitmap (definite punto per punto, cioè pixel per pixel). E' questa la classe che sovraintende a tutte le operazioni di disegno in Java mediante i suoi metodi.

Il **sistema di coordinate** di Java ha l'origine nel vertice in alto a sinistra del componente su cui si disegna, con valori positivi dell'ascissa x verso destra e dell'ordinata y verso il basso, come è rappresentato in figura 9.1.

Le coordinate vengono espresse in pixel, cioè in numero di punti sullo schermo, mediante numeri interi.

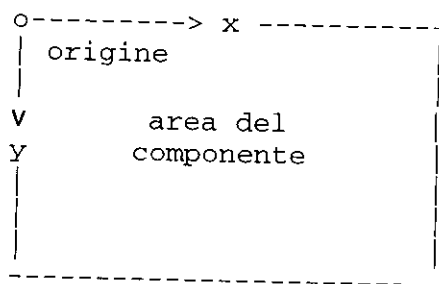



Figura 9.1 Sistema di coordinate in Java

Il **colore** con cui vengono effettuati i disegni è quello stabilito eventualmente con un'istruzione `g.setColor(colore);` oppure quello di default (cioè il nero) nel caso che non ne sia stato precisato alcuno.

2.1.3 Linee

Per disegnare un segmento di retta tra due punti la classe `Graphics` dispone del metodo **`drawLine()`**, che richiede come argomenti le coordinate dei due punti:

```
drawLine(int x1, int y1, int x2, int y2)
```

Le linee tracciate con questo metodo, così come quelle tracciate dagli altri metodi per disegnare altri elementi geometrici, sono sempre di un solo pixel di spessore (nella prima versione di Java non c'è la possibilità di cambiare lo spessore delle linee disegnate).

Esempio pratico in cui viene tracciato un segmento tra il punto (0, 0), cioè il vertice in alto a sinistra, e il punto (100, 100) spostato 100 pixel a destra e 100 verso il basso:

```
// I01linea.java (F.Spagna) Esempio di disegno di un segmento tra due punti

import java.awt.*;

public class I01linea extends java.applet.Applet {

   public void paint(Graphics g) {
```

```
        g.drawLine(0, 0, 100, 100);
    }
}
```

In figura 9.16 è riportato il risultato grafico dell'applet dell'esempio.

Il metodo **drawPolyline(int x[], nt y[], int nPunti)** traccia una sequenza di segmenti che uniscono una serie di punti definiti dagli array di coordinate x[] e y[]: la spezzata non viene chiusa alla fine se il primo e l'utltimo punto non coincidono.

Esempio#:

2.1.4 Rettangoli

Per disegnare un rettangolo la classe `Graphics` dispone del metodo **drawRect()**, che richiede come argomenti le coordinate x e y del vertice in alto a sinistra e la larghezza e l'altezza del rettangolo:

```
drawRect(int x, int y, int larghezza, int altezza)
```

Il metodo **fillRect()** è del tutto analogo, con la differenza che viene riempito con il colore corrente del disegno.

Per disegnare rettangoli speciali esistono ancora altri metodi come **drawRoundRect()** per disegnare rettangoli con i vertici arrotondati secondo un quarto di ellisse e il corrispondente **fillRoundRect()** che ne fa anche il riempimento con il colore corrente: questi metodi richiedono due argomenti supplementari che rappresentano la larghezza e l'altezza totale delle zone arrotondate (somma delle due parti lungo lo stesso lato) vicino ai vertici (vedi l'esempio che segue).

Un altro tipo di rettangolo che si può disegnare è quello prodotto dal metodo **draw3DRect()** che presenta un effetto tridimensionale molto rudimentale prodotto da due bordi in luce più chiari e due bordi in ombra più scuri. Sono possibili due effetti, quello che fa apparire il rettangolo in rilievo (come un bottone sollevato) e quello che lo fa apparire incassato (come un bottone premuto): queste due possibilità sono governate da un quinto parametro booleano che se è `true` produce un rettangolo in rilievo, se invece è `false` il rettangolo apparirà incassato.

Il metodo **clearRect()** riempie di colore il rettangolo come `fillRect()`, usando però il colore di fondo corrente anziché quello di disegno (è come se il fondo originario venisse ad essere scoperto se era occupato da precedenti disegni).

Ecco un esempio che comprende diversi rettangoli disegnati con i vari metodi:

```
// I02rettangoli.java  F.Spagna  Disegno di rettangoli
import java.awt.*;
```

```
public class I02rettangoli extends java.applet.Applet {

   public void paint(Graphics g) {

      g.drawRect(20, 25, 60, 50);
      g.fillRect(95, 25, 60, 50);
      g.drawRoundRect(170, 25, 60, 50, 30, 25);
      g.fillRoundRect(245, 25, 60, 50, 30, 25);
      g.setColor(Color.red);
      g.draw3DRect(20, 110, 60, 50, true);
      g.draw3DRect(95, 110, 60, 50, false);
      g.fill3DRect(170, 110, 60, 50, true);
      g.fill3DRect(245, 110, 60, 50, false);
   }
}
```

Il risultato è quello di figura 9.2. L'effetto di trimensionalità non è sempre evidente dato lo spessore del disegno dei bordi, che, essendo sempre di un pixel, è molto sottile.

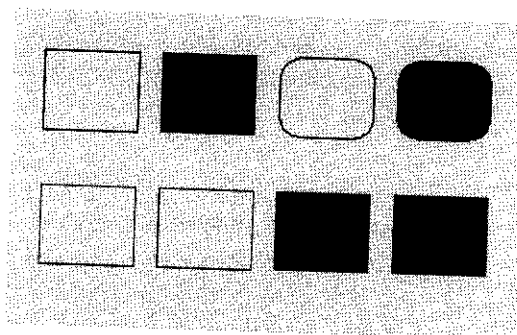



Figura 9.2 Disegno di rettangoli vuoti e pieni in vario modo.

2.1.5 Poligoni

2.1.5.1 Classe **Polygon** e creazione di un poligono

La classe **Polygon** del package # permette di creare un oggetto che rappresenta un poligono definito mediante l'insieme delle coordinate dei suoi vertici.

La creazione di un poligono può essere fatta o con un costruttore che ricevere i due array di ascisse e di ordinate dei vertici, oppure con un costruttore senza argomenti e applicando poi ripetutamente per ogni vertice il metodo `addPoint(x, y)`.

Per esempio la creazione di due poligoni rappresentanti una lettera T con altezza di 200 pixel, larghezza di 150 pixel e spessore di 30 pixel ed inizializzati nei due modi detti sopra, si può fare nei due modi mostrati dell'esempio del paragrafo seguente.

2.1.5.2 Disegno di un poligono

Per disegnare un poligono può essere usato il metodo **drawPolygon()** di `Graphics` che può ricevere o l'una o l'altra delle seguenti liste di argomenti:

- un array di ascisse `x[]`, un array di ordinate `y[]` e il numero di vertici,

- un oggetto di tipo `Polygon` già opportunamente istanziato nel modo spiegato nel paragrafo precedente.

Esiste anche una versione **fillPolygon()** per disegnare il poligono riempito con il colore corrente.

Con il metodo `drawPolygon()` il poligono non viene chiuso automaticamente se il primo e l'ultimo vertice non coincidono: per farlo chiudere bisogna aggiungere ai suoi vertici un ultimo punto uguale al primo. Questo non è necessario con il metodo `fillPolygon()` che fa la chiusura automaticamente (e non potrebbe essere altrimenti dato che deve essere fatto il riempimento).

Nell'esempio seguente sono disegnati due poligoni, uno vuoto (che non è chiuso) ed uno pieno, e sono stati adoperati i due modi di definizione dei vertici: nel primo caso passando l'array dei vertici come argomento del costruttore e nel secondo caso adoperando ripetutamente il metodo `add()` (ci si perdoni la piccola operazione di inversione fatta sulle ordinate per rovesciare la figura).

```
// I03poligoni.java (F.Spagna) Disegno di poligoni

import java.awt.*;

public class I03poligoni extends java.applet.Applet {

   int x[] = { 20, 170, 170, 110, 110,  80, 80, 20};
   int y[] = { 20,  20,  50,  50, 170, 170, 50, 50};
   Polygon Tvuoto = new Polygon(x, y, x.length);
   Polygon Tpieno = new Polygon();

   public void init() {
      for (int n = 0; n < x.length; n++)
         Tpieno.addPoint(x[n]+130, 200 - y[n]);
   }
   public void paint(Graphics g) {
      g.drawPolygon(Tvuoto);
      g.fillPolygon(Tpieno);
   }
}
```

Il risultato del disegno è riportato in figura 9.3.

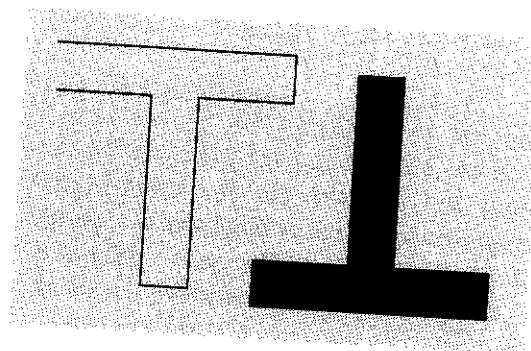



Figura 9.3 Disegno di un poligono vuoto e di uno pieno.

2.1.6 Ellissi o cerchi

Un'ellisse viene disegnata con il metodo **drawOval()** di `Graphics` che riceve gli argomenti che si dovrebbero dare per il disegno del rettangolo in cui l'ellisse è inscritta Per ben fissare nella memoria gli argomenti che si devono passare al metodo facciamo proprio un esempio disegnando un'ellisse ma anche per riferimento il rettangolo in cui essa è inscritta. Analogamente ad altri casi visti precedentemente, questo metodo ha una versione **fillOval()** per figure piene, che riportiamo pure nell'esempio.

```
// I04ellissi.java  F.Spagna  Disegno di ellissi
import java.awt.*;

public class I04ellissi extends java.applet.Applet {

   public void paint(Graphics g) {

      g.drawOval(20, 20, 120, 70);
      g.drawRect(20, 20, 120, 70);
      g.fillOval(160, 20, 120, 70);
      g.drawRect(160, 20, 120, 70);
   }
}
```

Il risultato è quello di figura 9.4.

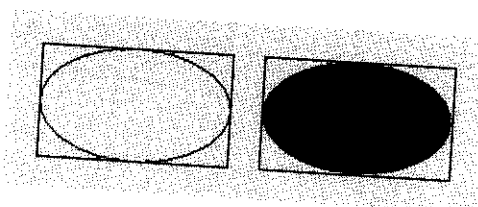



Figura 9.4 Disegno di un'ellisse vuota e di una piena.

2.1.7 Archi di ellisse o di cerchio

Il disegno di archi di ellisse e quindi, come caso particolare, anche degli archi di cerchio, può essere fatto con il metodo **drawArc()** di Graphics. Per quanto riguarda gli argomenti richiesti da questo metodo bisogna ricordare innanzi tutto i quattro argomenti richiesti dal metodo drawOval(), che per gli archi di ellisse sono definiti nello stesso modo con riferimento all'ellisse completa di cui l'arco fa parte (cioè le coordinate del vertice in alto a sinistra e base e altezza del rettangolo in cui si può inscrivere l'ellisse). Ci sono poi da aggiungere due altri parametri che indicano il primo l'angolo di inizio dell'arco rispetto all'asse orizzontale dell'ellisse a partire dalla posizione di una lancetta di orologio indicante le ore tre, e il secondo l'angolo da percorrere per arrivare alla fine dell'arco (apertura dell'arco), essendo l'angolo iniziale e l'apertura dell'angolo considerati positivi andando in senso antiorario e gli angoli essendo espressi in gradi (sono accettati anche valori negativi che vanno in senso orario).

Il metodo **fillArc()** segue le stesse regole per gli argomenti, salvo che riempie del colore corrente tutto lo spicchio di ellisse relativo all'arco (a guisa di una fetta di torta). Facciamo un esempio, in cui sono disegnati, per fissare meglio le idee, anche i rettangoli in cui l'ellisse è inscritta:

```
// I05archi.java  F.Spagna  Disegno di archi di ellisse

import java.awt.*;

public class I05archi extends java.applet.Applet {

   public void paint(Graphics g) {

      g.drawArc(20, 20, 120, 70, 0, 90);
      g.drawRect(20, 20, 120, 70);
      g.fillArc(160, 20, 120, 70, 0, 90);
      g.drawRect(160, 20, 120, 70);
      g.drawArc(20, 110, 120, 70, 90, 270);
      g.drawRect(20, 110, 120, 70);
      g.fillArc(160, 110, 120, 70, 90, 270);
      g.drawRect(160, 110, 120, 70);
   }
}
```

Il risultato è riportato in figura 9.5.

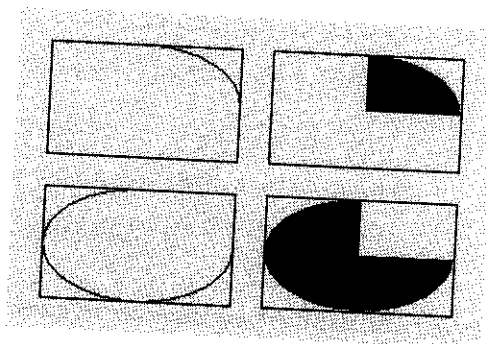



Figura 9.5 Disegno di archi di ellisse vuoti e pieni (in alto partenza da 0° e angolo di apertura di 90° e in basso partenza da 90° e angolo di apertura di 270°).

2.1.8 Scrittura di stringhe

Per scrivere una stringa su un componente grafico si può usare, oltre che un componente di tipo `Label` (a questo proposito vedi il paragrafo 7.2.1), anche il metodo **`drawString()`** di `Graphics`. Ma prima, se non ci si vuole limitare ad usare il font che il sistema adotta di default, bisogna fissare il font scelto per le scritte con il metodo `setFont(Font)` del componente sul quale si deve scrivere, oppure lo stesso metodo, ma del suo contesto grafico `Graphics`.

Per esempio:

```
Font f = new Font("TimesRoman", Font.PLAIN, 14);
g.setFont(f);
```

o, più sinteticamente:

```
g.setFont(new Font("TimesRoman", Font.PLAIN, 14));
```

Il metodo `drawString()` riceve come argomenti la stringa da scrivere e le coordinate x e y del punto da dove comincia la scritta, con y riferito alla linea base della scritta stessa.

Nella figura seguente 9.6 è tracciato uno schema relativo alle suddette coordinate.

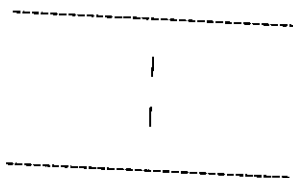



Figura 9.6 Coordinate relative al disegno di una stringa su un componente.

Si riporta qui un esempio che utilizza in una serie di scritte vari font. Il risultato è presentato in figura 9.7.

```
// I06stringhe.java (F.Spagna) Disegno di archi di ellisse

import java.awt.*;

public class I06stringhe extends java.applet.Applet {

   public void paint(Graphics g) {

      for (int i = 0; i < 12; i++) {
         g.setFont(new Font ("TimesRoman", Font.PLAIN, 10+2*i));
         g.drawString("TimesRoman"+(10+2*i), 20, 30 + 25*i);
      }
      for (int i = 0; i < 12; i++) {
         g.setFont(new Font ("TimesRoman", Font.BOLD, 10+2*i));
         g.drawString("bold", 225, 30 + 25*i);
      }
      for (int i = 0; i < 12; i++) {
         g.setFont(new Font ("TimesRoman", Font.ITALIC, 10+2*i));
         g.drawString("italic", 295, 30 + 25*i);
      }
      for (int i = 0; i < 12; i++) {
         g.setFont(new Font ("Courier", Font.PLAIN, 10+2*i));
         g.drawString("Courier", 372, 30 + 25*i);
      }
      for (int i = 0; i < 12; i++) {
         g.setFont(new Font ("Helvetica", Font.PLAIN, 10+2*i));
         g.drawString("Helvetica", 513, 30 + 25*i);
      }
   }
}
```

TimesRoman10 bold *italic* Courier Helvetica
TimesRoman12 bold *italic* Courier Helvetica
TimesRoman14 bold *italic* Courier Helvetica
TimesRoman16 bold *italic* Courier Helvetica
TimesRoman18 bold *italic* Courier Helvetica
TimesRoman20 bold *italic* Courier Helvetica
TimesRoman22 bold *italic* Courier Helvetica
TimesRoman24 bold *italic* Courier Helvetica
TimesRoman26 bold *italic* Courier Helvetica
TimesRoman28 bold *italic* Courier Helvetica
TimesRoman30 bold *italic* Courier Helvetica
TimesRoman32 bold *italic* Courier Helvetica

Figura 9.7 Scritte di stringhe con vari tipi di font e dimensioni diverse.

Un metodo alternativo per scrivere una serie di caratteri sullo schermo è il **drawChars()** che riceve però come argomenti non una stringa bensì un array di caratteri, che è un tipo di oggetto diverso, ed ha ancora altri due argomenti con i quali si possono precisare gli indici all'interno dell'array del primo e dell'ultimo carattere da scrivere, essendo scritti tutti i caratteri compresi tra i due.

Esempio:

#

2.1.9 Immagini

2.1.9.1 Classe **Image**

La classe **Image** del package `java.awt` permette di creare ed usare oggetti che rappresentano un'immagine e dispongono di vari metodi per il suo trattamento.

La classe `Applet` presenta un metodo per caricare un'immagine e la classe `Graphics` ha un metodo per disegnare un'immagine sullo schermo.

2.1.9.2 Caricamento di un'immagine

Prima di essere utilizzata in un'applet un'immagine deve essere caricata attraverso la rete come file a partire dal server dell'applet mediante il metodo **getImage()** della classe `Applet` che carica sulla macchina client l'immagine da un determinato server e ne crea un'istanza disponibile per il programma. Il metodo può ricevere come argomento l'URL dell'immagine (come oggetto di tipo `URL`), comprendente la directory ed il nome del file immagine:

```
Image img = getImage(URL);
```

Un esempio potrebbe essere:

```
Image img = getImage(new URL(http://www.server.it/immagini/immagine.gif));
```

Ma una forma alternativa del metodo, che è preferibile perchè più flessibile, prevede due argomenti separati, uno per l'URL del server (oggetto di tipo `URL`) e l'altro per il nome del file di immagine preceduto dalla directory all'interno del server relativa a quella contenente il documento HTML (una stringa).

Per definire l'URL del file HTML che la richiama, un'applet ha, in quanto appartenente alla classe `Applet`, a disposizione il metodo `getDocumentBase()` che restituisce l'URL della directory contenente il documento HTML come oggetto di tipo `URL`, ma dispone anche del metodo `getCodeBase()` che restituisce l'URL della directory contenente invece l'applet, che può in generale essere diversa da quella dell'HTML (ciò quando è precisato un `codebase` nel tag `<applet>`, secondo quanto detto al paragrafo 5.4.2). L'uso dell'uno o dell'altro metodo dipende dalla localizzazione del file immagine (nella directory del file HTML che richiama l'applet o in quella del file `.class` costituente l'applet stessa). Qundi nei due casi:

```
Image img = getImage(getDocumentBase(), "immagine.gif");
```

oppure:

```
Image img = getImage(getCodeBase(), "immagine.gif");
```

Nel caso di animazioni che comprendono molte immagini di solito queste vengono poste in una sottodirectory a parte sotto la directory contenente l'applet. Se ad esempio una tale sottodirectory si chiamasse `immagini`, si scriverebbe:

```
Image img = getImage(getCodeBase(), "immagini/immagine.gif");
```

Se il programma richiede un'immagine e non la trova esso non ne fa un dramma e viene eseguito lo stesso, ma senza evidentemente che l'immagine possa essere vista.

I formati di immagini accettati da Java al momento sono solo il GIF ed il JPEG, che sono poi quelli attualmente riconosciuti dai browser per Internet.

Fuori da un'applet un'immagine può essere caricata anche con:

```
Image img = Toolkit.getDefaultToolkit().getImage(nomefilenaOppureURL);
```

Esempio:

```
// I00toolkit.java F.Spagna Immagine caricata in un'applicazione Java

import java.awt.*;
import java.awt.event.*;

public class I00toolkit extends Frame {

   static Image mappa;                           // immagine mappa da zoomare

   I00toolkit() {
      mappa = Toolkit.getDefaultToolkit().getImage("Italia.gif");
   }
   public void paint(Graphics g) {
      g.drawImage(mappa, 0, 0, this);             // disegna l'immagine reale
   }
   public static void main(String s[]) {

      I00toolkit fr = new I00toolkit();
      fr.pack();
      fr.setVisible(true);
   }
}
```

2.1.9.3 Riproduzione di un'immagine

Per riprodurre un'immagine su un componente si può utilizzare il metodo **`drawImage()`** della classe `Graphics`, avente come argomenti l'immagine stessa come oggetto di tipo `Image`,

le coordinate x e y del vertice in alto a sinistra (come quelle del disegno di un rettangolo, per intenderci) e la referenza del componente stesso (questa espressa con `this`). Ad esempio:

```
g.drawImage(img, 10, 10, this);
```

Nel caso nel disegno si vogliano dare all'immagine delle dimensioni diverse da quelle originali si può ricorrere ad un metodo *overloaded* del precedente che prevede altri due argomenti, che sono la nuova larghezza e la nuova altezza dell'immagine da disegnare, assegnati indipendentemente dalle dimensioni originali:

```
g.drawImage(img, x, y, largh, alt, this);
```

Se nel cambiare le dimensioni di un'immagine si vogliono riferire le dimensioni a quelle originali si possono utilizzare i metodi `getWidth(this)` e `getHeight(this)` della classe `Image`, che rilevano rispettivamente la larghezza e l'altezza dell'immagine originale, e poi rapportare le dimensioni del disegno ad esse (vedi l'esempio che segue). Il ridimensionamento dell'immagine, sia riducendola sia ingrandendola, può produrre un peggioramento della sua qualità. Se poi i due lati sono modificati in misura diversa, cioà se il rapporto dei lati è diverso da quello originario, l'immagine risulta evidentemente deformata.

A proposito dell'argomento di `getWidth(this)` e `getHeight(this)` diciamo qui soltanto che il parametro che i due metodi si aspettano è un oggetto di tipo `ImageObserver`, che cioè implementa quell'interfaccia che ha a che fare con il processo di caricamento dell'immagine (permette di conoscere a che punto si è nel processo di caricamento) e che il più delle volte può essere rappresentato dal `this`, cioè il componente stesso su cui si disegna. La stessa cosa vale per il `this` adoperato come ultimo parametro nel metodo `drawImage()` della classe `Graphics`.

Esempio in cui un'immagine GIF viene riprodotta in un'applet una volta in grandezza reale e una seconda volta anche con dimensioni ridotte della metà.

```
// I07immagine.java (F.Spagna)Riproduzione di immagini in scala reale e ridotta
import java.awt.*;

public class I07immagine extends java.applet.Applet {

   Image im;
   public void init() {
      im = getImage(getCodeBase(), "aereo.gif");
   }
   public void paint(Graphics g) {
      g.drawImage(im, 10, 10, this);
      g.drawImage(im,220,150,img.getWidth(this)/2,img.getHeight(this)/2,this);
   }
}
```

Il risultato è riprodotto in figura 9.8.

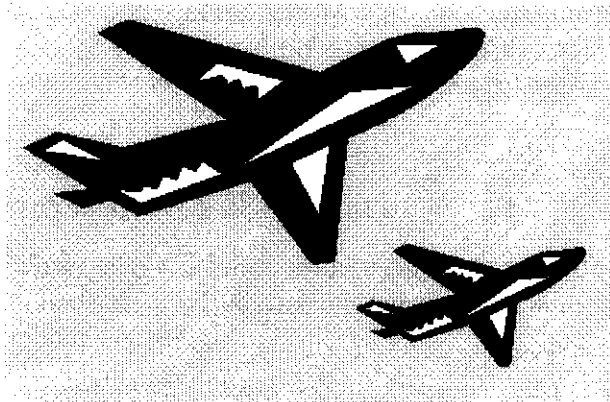



Figura 9.8 Disegno su un'applet di un'immagine normale ed una ridotta.

2.1.9.4 Trattamento delle immagini

Il package `java.awt.Image` (da non confondere con il package `java.awt` che contiene la classe `Image`) contiene delle classi e delle interfacce utili per il trattamento delle immagini mediante operazioni sui bit e sui colori.

2.1.10 Copia di un'area di disegno

Mediante il metodo **`copyArea()`** della classe `Graphics` è possibile copiare con il suo contenuto grafico da un posto ad un altro dell'area di disegno un'intera area rettangolare già disegnata. La lista dei parametri da passare al metodo comprende i quattro soliti parametri di un rettangolo, che è quello dell'area da copiare (le coordinate del vertice in alto a sinistra, la larghezza e l'altezza) e in più lo spostamento orizzontale e lo spostamento verticale rispetto alla posizione originale.

```
g.copyArea(int x, int y, int bas, int alt, int dx, int dy);
```

2.1.11 I colori in Java

2.1.11.1 La classe `Color`

La classe **`Color`** serve per gestire il colore in Java. Un determinato colore può essere rappresentato da un oggetto di questa classe. Java prende in considerazione colori definiti con 24 bit, cioè con 8 bit per ciascuna delle tre componenti fondamentali del colore (rosso, verde e blù), con valori quindi che possono andare da 0 a 255 per ogni componente e la conseguente possibilità di trattare un numero totale di 256^3 colori (oltre 16 milioni di colori).

Però il sistema che ospita l'applicazione o l'applet e i browser al momento generalmente non supportano che un valore molto inferiore di colori (spesso anche solo 256) e allora viene fatta una

mappatura di corrispondenze di colori su un numero di colori molto più limitato di quelli teorici, oppure viene fatto un *dithering* (colori composti con una mescolanza di singoli pixel di colori diversi) secondo la scelta che è stata fatta nel sistema operativo e nel browser utilizzato.

2.1.11.2 Colori predefiniti e altri colori

Nella classe `Color` c'è un certo numero di colori, quelli più comuni, predefiniti come membri di classe (variabili statiche costanti), e quindi usabili direttamente in nome della classe senza dover istanziare uno specifico oggetto, che sono:

Color.black, Color.blue, Color.cyan, Color.darkGray, Color.gray, Color.green, Color.lightGray, Color.magenta, Color.orange, Color.pink, Color.red, Color.white, Color.yellow

Per altri colori si può istanziare un oggetto della classe `Color` inizializzato in vario modo, per esempio con i valori interi (compresi tra 0 e 255) delle tre componenti fondamentali (rosso, verde e blù) del colore voluto.

```
Color c = new Color(int red, int green, int blue);
```

Esiste anche un costruttore alternativo che riceve le tre componenti come numeri in virgola mobile compresi tra 0.0 e 1.0.

Nel package `java.awt` sono anche definiti dei colori in relazione a quelli del browser:(vedi#)

editableText, menuBack, menuBright, menuDim, menuFore, menuHighlight, readOnlyText.

Esempio:#

La classe **`SystemColor`** dell'AWT, derivata da **`Color`**, permette di rilevare i colori dei vari componenti grafici dell'interfaccia utente del sistema in uso, attraverso i valori di una serie (25 in tutto) di variabili statiche di classe che restituiscono i colori in termini di istanze della classe **`Color`**.

Il programma seguente (un'applet) va a vedere i vari colori per un sistema Windows (con colori standard) e ne presenta per ognuno un'etichetta di quel colore sull'applet.

```
// I08colori.java (F.Spagna) Colori dei vari componenti di sistema

import java.awt.*;

public class I08colori extends java.applet.Applet {

   static Color colore[] = {
      SystemColor.desktop,
      SystemColor.activeCaption,
      SystemColor.activeCaptionText,
      SystemColor.activeCaptionBorder,
      SystemColor.inactiveCaption,
```

```
      SystemColor.inactiveCaptionText,
      SystemColor.inactiveCaptionBorder,
      SystemColor.window,
      SystemColor.windowBorder,
      SystemColor.windowText,
      SystemColor.menu,
      SystemColor.menuText,
      SystemColor.text,
      SystemColor.textHighlight,
      SystemColor.textHighlightText,
      SystemColor.textInactiveText,
      SystemColor.textText,
      SystemColor.control,
      SystemColor.controlText,
      SystemColor.controlHighlight,
      SystemColor.controlLtHighlight,
      SystemColor.controlShadow,
      SystemColor.controlDkShadow,
      SystemColor.info,
      SystemColor.infoText
   };
   static String elem[] = {
      "desktop",
      "activeCaption",
      "activeCaptionText",
      "activeCaptionBorder",
      "inactiveCaption",
      "inactiveCaptionText",
      "inactiveCaptionBorder",
      "window",
      "windowBorder",
      "windowText",
      "menu",
      "menuText",
      "text",
      "textHighlight",
      "textHighlightText",
      "textInactiveText",
      "textText",
      "control",
      "controlText",
      "controlHighlight",
      "controlLtHighlight",
      "controlShadow",
      "controlDkShadow",
      "info",
      "infoText"
   };
   public void init() {
      setLayout(new GridLayout(colore.length, 2));
      for (int n = 0; n < colore.length; n++) {
         Label l1 = new Label("" + n + ": " + elem[n]);
         add(l1);
         Label l2 = new Label("     ");
         l2.setBackground(colore[n]);
         add(l2);
      }
   }
}
```

In figura 9.9 si può vedere il risultato di una tale applet.

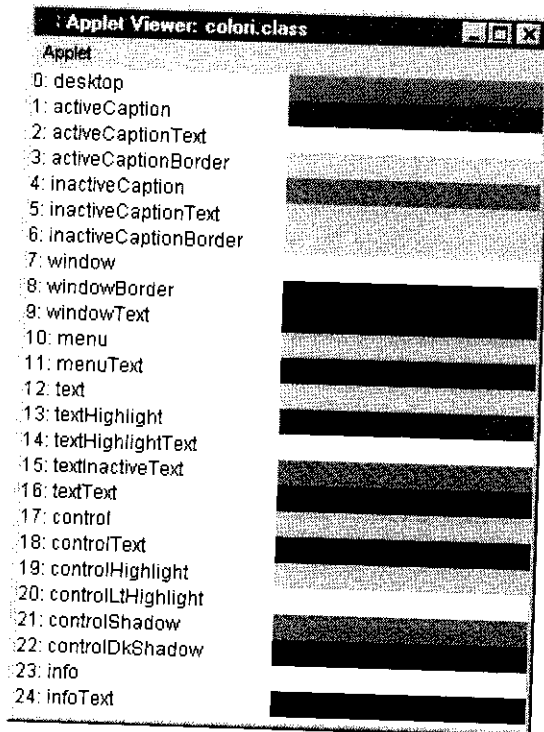



Figura 9.9 Applet che rileva i colori del sistema.

2.1.11.3 Colori correnti in un contesto grafico e in un componente

La classe `Graphics` prevede un metodo **`setColor()`** per stabilire il colore corrente di ogni disegno (figure geometriche) o scritta (stringhe) che venga fatto con quel contesto grafico. Per esempio con:

```
g.setColor(Color.red);
```

tutti i disegni e le scritte che seguono questa istruzione sono fatti in rosso.

Ma si può anche fissare il colore del disegno (foreground) e il colore di fondo (background) per il componente (oggetto di classe `Component`) su cui si disegna, con i metodi del componente (cioè della sua classe) **`setForeground(Color)`** e **`setBackground(Color)`** (vedi anche il paragrafo 7.1.5). Si osservi a questo proposito che, dato un componente (oggetto) su cui si fanno delle operazioni grafiche, si possono fissare le caratteristiche di disegno, come anche quelle del font, sul componente stesso (in un'applet generalmente nel suo metodo `init()`) oppure assegnare le specificazioni grafiche all'oggetto `Graphics` che viene passato al componente nel metodo `paint(Graphics)`. Tra i due sistemi prevale quello che assegna il colore di disegno al componente, così che per esempio si può cambiare di colpo il colore di ogni cosa disegnata sul componente con una sola istruzione di tipo `setForeground(Color)`.

Esempio:#

Per default il colore di fondo di ogni componente è il grigio in una tonalità media.

2.1.11.4 Operazioni sui colori

Se si vuole creare una gamma di colori tutti con la stessa saturazione e la stessa luminosità, ma con tinta diversa dall'uno all'altro basta variare il parametro `hue` nel metodo seguente, lasciando costanti gli altri due:(vedi#)

```
getHSBColor(hue, saturation, brightness)
```

Esempio:

```
// I00hue.java F.Spagna Colori con hue variabile


import java.awt.*;


public class I00hue extends java.applet.Applet {


   public void paint(Graphics g) {
      for (int i = 0; i < 256; i++)
      for (int n = 0; n < 256; n++) {
         g.setColor(Color.getHSBColor((float)(n/256.), (float)(i/256.), 0.8f));
         g.drawLine(n, 2*i, n, 2*i+1);
      }
   }
}
```

2.1.11.5 Una tabella di colori

Riportiamo qui una tabella di colori che è presente nei sistemi UNIX sotto forma di un file chiamato `rgb.tx`, pensando che possa essere di qualche utilità quando si lavori con i colori.

```
255 250 250    snow

248 248 255    ghost white

248 248 255    GhostWhite

245 245 245    white smoke
245 245 245    WhiteSmoke
220 220 220    gainsboro
255 250 240    floral white
255 250 240    FloralWhite
```

```
253 245 230     old lace
253 245 230     OldLace
250 240 230     linen
250 235 215     antique white
250 235 215     AntiqueWhite
255 239 213     papaya whip
255 239 213     PapayaWhip
255 235 205     blanched almond
255 235 205     BlanchedAlmond
255 228 196     bisque
255 218 185     peach puff
255 218 185     PeachPuff
255 222 173     navajo white
255 222 173     NavajoWhite
255 228 181     moccasin
255 248 220     cornsilk
255 255 240     ivory
255 250 205     lemon chiffon
255 250 205     LemonChiffon
255 245 238     seashell
240 255 240     honeydew
245 255 250     mint cream
245 255 250     MintCream
240 255 255     azure
240 248 255     alice blue
240 248 255     AliceBlue
230 230 250     lavender
255 240 245     lavender blush
255 240 245     LavenderBlush
255 228 225     misty rose
255 228 225     MistyRose
255 255 255     white
  0   0   0     black
 47  79  79     dark slate gray
 47  79  79     DarkSlateGray
 47  79  79     dark slate grey
 47  79  79     DarkSlateGrey
105 105 105     dim gray
105 105 105     DimGray
105 105 105     dim grey
105 105 105     DimGrey
112 128 144     slate gray
112 128 144     SlateGray
112 128 144     slate grey
112 128 144     SlateGrey
119 136 153     light slate gray
119 136 153     LightSlateGray
119 136 153     light slate grey
119 136 153     LightSlateGrey
190 190 190     gray
190 190 190     grey
211 211 211     light grey
211 211 211     LightGrey
211 211 211     light gray
211 211 211     LightGray
 25  25 112     midnight blue
 25  25 112     MidnightBlue
  0   0 128     navy
  0   0 128     navy blue
  0   0 128     NavyBlue
100 149 237     cornflower blue
100 149 237     CornflowerBlue
 72  61 139     dark slate blue
 72  61 139     DarkSlateBlue
106  90 205     slate blue
106  90 205     SlateBlue
123 104 238     medium slate blue
123 104 238     MediumSlateBlue
132 112 255     light slate blue
132 112 255     LightSlateBlue
  0   0 205     medium blue
  0   0 205     MediumBlue
 65 105 225     royal blue
 65 105 225     RoyalBlue
```

```
  0   0 255   blue
 30 144 255   dodger blue
 30 144 255   DodgerBlue
  0 191 255   deep sky blue
  0 191 255   DeepSkyBlue
135 206 235   sky blue
135 206 235   SkyBlue
135 206 250   light sky blue
135 206 250   LightSkyBlue
 70 130 180   steel blue
 70 130 180   SteelBlue
176 196 222   light steel blue
176 196 222   LightSteelBlue
173 216 230   light blue
173 216 230   LightBlue
176 224 230   powder blue
176 224 230   PowderBlue
175 238 238   pale turquoise
175 238 238   PaleTurquoise
  0 206 209   dark turquoise
  0 206 209   DarkTurquoise
 72 209 204   medium turquoise
 72 209 204   MediumTurquoise
 64 224 208   turquoise
  0 255 255   cyan
224 255 255   light cyan
224 255 255   LightCyan
 95 158 160   cadet blue
 95 158 160   CadetBlue
102 205 170   medium aquamarine
102 205 170   MediumAquamarine
127 255 212   aquamarine
  0 100   0   dark green
  0 100   0   DarkGreen
 85 107  47   dark olive green
 85 107  47   DarkOliveGreen
143 188 143   dark sea green
143 188 143   DarkSeaGreen
 46 139  87   sea green
 46 139  87   SeaGreen
 60 179 113   medium sea green
 60 179 113   MediumSeaGreen
 32 178 170   light sea green
 32 178 170   LightSeaGreen
152 251 152   pale green
152 251 152   PaleGreen
  0 255 127   spring green
  0 255 127   SpringGreen
124 252   0   lawn green
124 252   0   LawnGreen
  0 255   0   green
127 255   0   chartreuse
  0 250 154   medium spring green
  0 250 154   MediumSpringGreen
173 255  47   green yellow
173 255  47   GreenYellow
 50 205  50   lime green
 50 205  50   LimeGreen
154 205  50   yellow green
154 205  50   YellowGreen
 34 139  34   forest green
 34 139  34   ForestGreen
107 142  35   olive drab
107 142  35   OliveDrab
189 183 107   dark khaki
189 183 107   DarkKhaki
240 230 140   khaki
238 232 170   pale goldenrod
238 232 170   PaleGoldenrod
250 250 210   light goldenrod yellow
250 250 210   LightGoldenrodYellow
255 255 224   light yellow
255 255 224   LightYellow
255 255   0   yellow
```

```
255 215   0    gold
238 221 130    light goldenrod
238 221 130    LightGoldenrod
218 165  32    goldenrod
184 134  11    dark goldenrod
184 134  11    DarkGoldenrod
188 143 143    rosy brown
188 143 143    RosyBrown
205  92  92    indian red
205  92  92    IndianRed
139  69  19    saddle brown
139  69  19    SaddleBrown
160  82  45    sienna
205 133  63    peru
222 184 135    burlywood
245 245 220    beige
245 222 179    wheat
244 164  96    sandy brown
244 164  96    SandyBrown
210 180 140    tan
210 105  30    chocolate
178  34  34    firebrick
165  42  42    brown
233 150 122    dark salmon
233 150 122    DarkSalmon
250 128 114    salmon
255 160 122    light salmon
255 160 122    LightSalmon
255 165   0    orange
255 140   0    dark orange
255 140   0    DarkOrange
255 127  80    coral
240 128 128    light coral
240 128 128    LightCoral
255  99  71    tomato
255  69   0    orange red
255  69   0    OrangeRed
255   0   0    red
255 105 180    hot pink
255 105 180    HotPink
255  20 147    deep pink
255  20 147    DeepPink
255 192 203    pink
255 182 193    light pink
255 182 193    LightPink
219 112 147    pale violet red
219 112 147    PaleVioletRed
176  48  96    maroon
199  21 133    medium violet red
199  21 133    MediumVioletRed
208  32 144    violet red
208  32 144    VioletRed
255   0 255    magenta
238 130 238    violet
221 160 221    plum
218 112 214    orchid
186  85 211    medium orchid
186  85 211    MediumOrchid
153  50 204    dark orchid
153  50 204    DarkOrchid
148   0 211    dark violet
148   0 211    DarkViolet
138  43 226    blue violet
138  43 226    BlueViolet
160  32 240    purple
147 112 219    medium purple
147 112 219    MediumPurple
216 191 216    thistle
255 250 250    snow1
238 233 233    snow2
205 201 201    snow3
139 137 137    snow4
255 245 238    seashell1
238 229 222    seashell2
```

```
205 197 191     seashell3
139 134 130     seashell4
255 239 219     AntiqueWhite1
238 223 204     AntiqueWhite2
205 192 176     AntiqueWhite3
139 131 120     AntiqueWhite4
255 228 196     bisque1
238 213 183     bisque2
205 183 158     bisque3
139 125 107     bisque4
255 218 185     PeachPuff1
238 203 173     PeachPuff2
205 175 149     PeachPuff3
139 119 101     PeachPuff4
255 222 173     NavajoWhite1
238 207 161     NavajoWhite2
205 179 139     NavajoWhite3
139 121  94     NavajoWhite4
255 250 205     LemonChiffon1
238 233 191     LemonChiffon2
205 201 165     LemonChiffon3
139 137 112     LemonChiffon4
255 248 220     cornsilk1
238 232 205     cornsilk2
205 200 177     cornsilk3
139 136 120     cornsilk4
255 255 240     ivory1
238 238 224     ivory2
205 205 193     ivory3
139 139 131     ivory4
240 255 240     honeydew1
224 238 224     honeydew2
193 205 193     honeydew3
131 139 131     honeydew4
255 240 245     LavenderBlush1
238 224 229     LavenderBlush2
205 193 197     LavenderBlush3
139 131 134     LavenderBlush4
255 228 225     MistyRose1
238 213 210     MistyRose2
205 183 181     MistyRose3
139 125 123     MistyRose4
240 255 255     azure1
224 238 238     azure2
193 205 205     azure3
131 139 139     azure4
131 111 255     SlateBlue1
122 103 238     SlateBlue2
105  89 205     SlateBlue3
 71  60 139     SlateBlue4
 72 118 255     RoyalBlue1
 67 110 238     RoyalBlue2
 58  95 205     RoyalBlue3
 39  64 139     RoyalBlue4
  0   0 255     blue1
  0   0 238     blue2
  0   0 205     blue3
  0   0 139     blue4
 30 144 255     DodgerBlue1
 28 134 238     DodgerBlue2
 24 116 205     DodgerBlue3
 16  78 139     DodgerBlue4
 99 184 255     SteelBlue1
 92 172 238     SteelBlue2
 79 148 205     SteelBlue3
 54 100 139     SteelBlue4
  0 191 255     DeepSkyBlue1
  0 178 238     DeepSkyBlue2
  0 154 205     DeepSkyBlue3
  0 104 139     DeepSkyBlue4
135 206 255     SkyBlue1
126 192 238     SkyBlue2
108 166 205     SkyBlue3
 74 112 139     SkyBlue4
```

```
176 226 255     LightSkyBlue1
164 211 238     LightSkyBlue2
141 182 205     LightSkyBlue3
 96 123 139     LightSkyBlue4
198 226 255     SlateGray1
185 211 238     SlateGray2
159 182 205     SlateGray3
108 123 139     SlateGray4
202 225 255     LightSteelBlue1
188 210 238     LightSteelBlue2
162 181 205     LightSteelBlue3
110 123 139     LightSteelBlue4
191 239 255     LightBlue1
178 223 238     LightBlue2
154 192 205     LightBlue3
104 131 139     LightBlue4
224 255 255     LightCyan1
209 238 238     LightCyan2
180 205 205     LightCyan3
122 139 139     LightCyan4
187 255 255     PaleTurquoise1
174 238 238     PaleTurquoise2
150 205 205     PaleTurquoise3
102 139 139     PaleTurquoise4
152 245 255     CadetBlue1
142 229 238     CadetBlue2
122 197 205     CadetBlue3
 83 134 139     CadetBlue4
  0 245 255     turquoise1
  0 229 238     turquoise2
  0 197 205     turquoise3
  0 134 139     turquoise4
  0 255 255     cyan1
  0 238 238     cyan2
  0 205 205     cyan3
  0 139 139     cyan4
151 255 255     DarkSlateGray1
141 238 238     DarkSlateGray2
121 205 205     DarkSlateGray3
 82 139 139     DarkSlateGray4
127 255 212     aquamarine1
118 238 198     aquamarine2
102 205 170     aquamarine3
 69 139 116     aquamarine4
193 255 193     DarkSeaGreen1
180 238 180     DarkSeaGreen2
155 205 155     DarkSeaGreen3
105 139 105     DarkSeaGreen4
 84 255 159     SeaGreen1
 78 238 148     SeaGreen2
 67 205 128     SeaGreen3
 46 139  87     SeaGreen4
154 255 154     PaleGreen1
144 238 144     PaleGreen2
124 205 124     PaleGreen3
 84 139  84     PaleGreen4
  0 255 127     SpringGreen1
  0 238 118     SpringGreen2
  0 205 102     SpringGreen3
  0 139  69     SpringGreen4
  0 255   0     green1
  0 238   0     green2
  0 205   0     green3
  0 139   0     green4
127 255   0     chartreuse1
118 238   0     chartreuse2
102 205   0     chartreuse3
 69 139   0     chartreuse4
192 255  62     OliveDrab1
179 238  58     OliveDrab2
154 205  50     OliveDrab3
105 139  34     OliveDrab4
202 255 112     DarkOliveGreen1
188 238 104     DarkOliveGreen2
```

```
162 205  90   DarkOliveGreen3
110 139  61   DarkOliveGreen4
255 246 143   khaki1
238 230 133   khaki2
205 198 115   khaki3
139 134  78   khaki4
255 236 139   LightGoldenrod1
238 220 130   LightGoldenrod2
205 190 112   LightGoldenrod3
139 129  76   LightGoldenrod4
255 255 224   LightYellow1
238 238 209   LightYellow2
205 205 180   LightYellow3
139 139 122   LightYellow4
255 255   0   yellow1
238 238   0   yellow2
205 205   0   yellow3
139 139   0   yellow4
255 215   0   gold1
238 201   0   gold2
205 173   0   gold3
139 117   0   gold4
255 193  37   goldenrod1
238 180  34   goldenrod2
205 155  29   goldenrod3
139 105  20   goldenrod4
255 185  15   DarkGoldenrod1
238 173  14   DarkGoldenrod2
205 149  12   DarkGoldenrod3
139 101   8   DarkGoldenrod4
255 193 193   RosyBrown1
238 180 180   RosyBrown2
205 155 155   RosyBrown3
139 105 105   RosyBrown4
255 106 106   IndianRed1
238  99  99   IndianRed2
205  85  85   IndianRed3
139  58  58   IndianRed4
255 130  71   sienna1
238 121  66   sienna2
205 104  57   sienna3
139  71  38   sienna4
255 211 155   burlywood1
238 197 145   burlywood2
205 170 125   burlywood3
139 115  85   burlywood4
255 231 186   wheat1
238 216 174   wheat2
205 186 150   wheat3
139 126 102   wheat4
255 165  79   tan1
238 154  73   tan2
205 133  63   tan3
139  90  43   tan4
255 127  36   chocolate1
238 118  33   chocolate2
205 102  29   chocolate3
139  69  19   chocolate4
255  48  48   firebrick1
238  44  44   firebrick2
205  38  38   firebrick3
139  26  26   firebrick4
255  64  64   brown1
238  59  59   brown2
205  51  51   brown3
139  35  35   brown4
255 140 105   salmon1
238 130  98   salmon2
205 112  84   salmon3
139  76  57   salmon4
255 160 122   LightSalmon1
238 149 114   LightSalmon2
205 129  98   LightSalmon3
139  87  66   LightSalmon4
```

```
255 165   0   orange1
238 154   0   orange2
205 133   0   orange3
139  90   0   orange4
255 127   0   DarkOrange1
238 118   0   DarkOrange2
205 102   0   DarkOrange3
139  69   0   DarkOrange4
255 114  86   coral1
238 106  80   coral2
205  91  69   coral3
139  62  47   coral4
255  99  71   tomato1
238  92  66   tomato2
205  79  57   tomato3
139  54  38   tomato4
255  69   0   OrangeRed1
238  64   0   OrangeRed2
205  55   0   OrangeRed3
139  37   0   OrangeRed4
255   0   0   red1
238   0   0   red2
205   0   0   red3
139   0   0   red4
255  20 147   DeepPink1
238  18 137   DeepPink2
205  16 118   DeepPink3
139  10  80   DeepPink4
255 110 180   HotPink1
238 106 167   HotPink2
205  96 144   HotPink3
139  58  98   HotPink4
255 181 197   pink1
238 169 184   pink2
205 145 158   pink3
139  99 108   pink4
255 174 185   LightPink1
238 162 173   LightPink2
205 140 149   LightPink3
139  95 101   LightPink4
255 130 171   PaleVioletRed1
238 121 159   PaleVioletRed2
205 104 137   PaleVioletRed3
139  71  93   PaleVioletRed4
255  52 179   maroon1
238  48 167   maroon2
205  41 144   maroon3
139  28  98   maroon4
255  62 150   VioletRed1
238  58 140   VioletRed2
205  50 120   VioletRed3
139  34  82   VioletRed4
255   0 255   magenta1
238   0 238   magenta2
205   0 205   magenta3
139   0 139   magenta4
255 131 250   orchid1
238 122 233   orchid2
205 105 201   orchid3
139  71 137   orchid4
255 187 255   plum1
238 174 238   plum2
205 150 205   plum3
139 102 139   plum4
224 102 255   MediumOrchid1
209  95 238   MediumOrchid2
180  82 205   MediumOrchid3
122  55 139   MediumOrchid4
191  62 255   DarkOrchid1
178  58 238   DarkOrchid2
154  50 205   DarkOrchid3
104  34 139   DarkOrchid4
155  48 255   purple1
145  44 238   purple2
```

```
125  38 205    purple3
 85  26 139    purple4
171 130 255    MediumPurple1
159 121 238    MediumPurple2
137 104 205    MediumPurple3
 93  71 139    MediumPurple4
255 225 255    thistle1
238 210 238    thistle2
205 181 205    thistle3
139 123 139    thistle4
  0   0   0    gray0
  0   0   0    grey0
  3   3   3    gray1
  3   3   3    grey1
  5   5   5    gray2
  5   5   5    grey2
  8   8   8    gray3
  8   8   8    grey3
 10  10  10    gray4
 10  10  10    grey4
 13  13  13    gray5
 13  13  13    grey5
 15  15  15    gray6
 15  15  15    grey6
 18  18  18    gray7
 18  18  18    grey7
 20  20  20    gray8
 20  20  20    grey8
 23  23  23    gray9
 23  23  23    grey9
 26  26  26    gray10
 26  26  26    grey10
 28  28  28    gray11
 28  28  28    grey11
 31  31  31    gray12
 31  31  31    grey12
 33  33  33    gray13
 33  33  33    grey13
 36  36  36    gray14
 36  36  36    grey14
 38  38  38    gray15
 38  38  38    grey15
 41  41  41    gray16
 41  41  41    grey16
 43  43  43    gray17
 43  43  43    grey17
 46  46  46    gray18
 46  46  46    grey18
 48  48  48    gray19
 48  48  48    grey19
 51  51  51    gray20
 51  51  51    grey20
 54  54  54    gray21
 54  54  54    grey21
 56  56  56    gray22
 56  56  56    grey22
 59  59  59    gray23
 59  59  59    grey23
 61  61  61    gray24
 61  61  61    grey24
 64  64  64    gray25
 64  64  64    grey25
 66  66  66    gray26
 66  66  66    grey26
 69  69  69    gray27
 69  69  69    grey27
 71  71  71    gray28
 71  71  71    grey28
 74  74  74    gray29
 74  74  74    grey29
 77  77  77    gray30
 77  77  77    grey30
 79  79  79    gray31
 79  79  79    grey31
```

```
 82  82  82    gray32
 82  82  82    grey32
 84  84  84    gray33
 84  84  84    grey33
 87  87  87    gray34
 87  87  87    grey34
 89  89  89    gray35
 89  89  89    grey35
 92  92  92    gray36
 92  92  92    grey36
 94  94  94    gray37
 94  94  94    grey37
 97  97  97    gray38
 97  97  97    grey38
 99  99  99    gray39
 99  99  99    grey39
102 102 102    gray40
102 102 102    grey40
105 105 105    gray41
105 105 105    grey41
107 107 107    gray42
107 107 107    grey42
110 110 110    gray43
110 110 110    grey43
112 112 112    gray44
112 112 112    grey44
115 115 115    gray45
115 115 115    grey45
117 117 117    gray46
117 117 117    grey46
120 120 120    gray47
120 120 120    grey47
122 122 122    gray48
122 122 122    grey48
125 125 125    gray49
125 125 125    grey49
127 127 127    gray50
127 127 127    grey50
130 130 130    gray51
130 130 130    grey51
133 133 133    gray52
133 133 133    grey52
135 135 135    gray53
135 135 135    grey53
138 138 138    gray54
138 138 138    grey54
140 140 140    gray55
140 140 140    grey55
143 143 143    gray56
143 143 143    grey56
145 145 145    gray57
145 145 145    grey57
148 148 148    gray58
148 148 148    grey58
150 150 150    gray59
150 150 150    grey59
153 153 153    gray60
153 153 153    grey60
156 156 156    gray61
156 156 156    grey61
158 158 158    gray62
158 158 158    grey62
161 161 161    gray63
161 161 161    grey63
163 163 163    gray64
163 163 163    grey64
166 166 166    gray65
166 166 166    grey65
168 168 168    gray66
168 168 168    grey66
171 171 171    gray67
171 171 171    grey67
173 173 173    gray68
173 173 173    grey68
```

```
176 176 176     gray69
176 176 176     grey69
179 179 179     gray70
179 179 179     grey70
181 181 181     gray71
181 181 181     grey71
184 184 184     gray72
184 184 184     grey72
186 186 186     gray73
186 186 186     grey73
189 189 189     gray74
189 189 189     grey74
191 191 191     gray75
191 191 191     grey75
194 194 194     gray76
194 194 194     grey76
196 196 196     gray77
196 196 196     grey77
199 199 199     gray78
199 199 199     grey78
201 201 201     gray79
201 201 201     grey79
204 204 204     gray80
204 204 204     grey80
207 207 207     gray81
207 207 207     grey81
209 209 209     gray82
209 209 209     grey82
212 212 212     gray83
212 212 212     grey83
214 214 214     gray84
214 214 214     grey84
217 217 217     gray85
217 217 217     grey85
219 219 219     gray86
219 219 219     grey86
222 222 222     gray87
222 222 222     grey87
224 224 224     gray88
224 224 224     grey88
227 227 227     gray89
227 227 227     grey89
229 229 229     gray90
229 229 229     grey90
232 232 232     gray91
232 232 232     grey91
235 235 235     gray92
235 235 235     grey92
237 237 237     gray93
237 237 237     grey93
240 240 240     gray94
240 240 240     grey94
242 242 242     gray95
242 242 242     grey95
245 245 245     gray96
245 245 245     grey96
247 247 247     gray97
247 247 247     grey97
250 250 250     gray98
250 250 250     grey98
252 252 252     gray99
252 252 252     grey99
255 255 255     gray100
255 255 255     grey100
```

2.1.12 I font in Java

2.1.12.1 La classe **`Font`**

La classe **`Font`** rappresenta le caratteristiche, quali il nome del tipo di carattere, lo stile (normale, grassetto o corsivo) e le dimensioni, di un font utilizzato per le scritte che sono fatte su un determinato componente grafico, e presenta dei metodi per le operazioni che possono essere fatte su di esso.

Per creare un'istanza della classe `Font` si usa il costruttore:

```
Font(nome, stile, dimensione)
```

dove:

- `nome` è una stringa che caratterizza il modello di font, come ad esempio `"TimesRoman"`, `"Helvetica"`, `"Courier"`;

- `stile` rappresenta lo stile che può essere fissato con uno dei valori `Font.PLAIN` (per caratteri normali), `Font.BOLD` (per caratteri in grassetto) o `Font.ITALIC` (per caratteri in corsivo), che sono delle costanti statiche intere definite nella classe `Font` stessa, potendo anche essere dato uno stile combinato `Font.BOLD + Font.ITALIC` per sommare le due caratteristiche di grassetto e corsivo contemporaneamente;

- `dimensione` è la dimensione del font che rappresenta, anche se non sempre, l'altezza dei caratteri.

Per vedere un esempio con la stampa di stringhe con vari font si può andare al paragrafo 8.1.8 e alla figura 6.11.

Se non si trova sul sistema un font corrispondente a quello richiesto, Java adotta un font di default (di solito il Courier). Comunque la classe `Toolkit` del packege `java.awt` ha un metodo, `getFontList()`, che fornisce una lista dei font disponibili sul sistema usato.

#vedi progr. In Grafica 2D!

2.1.12.2 Caratteristiche di un font

La classe `Font` dispone di vari metodi per ricavare le caratteristiche di un font:

Metodo	Descrizione
`getName()`	restituisce il nome del font come stringa
`getStyle()`	restituisce lo stile del font (0 normale, 1 grassetto, 2 corsivo, 3 gras+cors)
`getSize()`	restituisce la dimensione del font
`isPlain()`	restituisce un booleano che dice se il font è normale
`isBold()`	restituisce un booleano che dice se il font è in grassetto
`isItalic()`	restituisce un booleano che dice se il font è in corsivo

Se si vuole conoscere il font utilizzato in un contesto grafico c'è il metodo `getFont()` della classe `Graphics`.

2.1.12.3 Dimensioni dei caratteri dei font e delle stringhe in pixel (classe **FontMetrics**)

Può essere utile a volte conoscere le caratteristiche dimensionali di un certo font usato, per esempio le dimensioni dei suoi caratteri per poter sistemare una scritta in un certo modo, ad esempio orizzontalmente e verticalmente al centro di una determinata area rettangolare. E' con la classe **FontMetrics** del package `java.awt` che si possono ottenere informazioni su queste caratteristiche, come l'altezza e la larghezza dei caratteri di un determinato font. Le dimensioni sono espresse in pixel.

Con `getFontMetrics(Font)` della classe `Component` o `getFontMetrics()` della classe `Graphics` si può istanziare un oggetto di tipo `FontMetrics`, che attraverso i suoi metodi fornisce le informazioni volute sulle dimensioni relative a quel font (si crea cioè un oggetto "misuratore" del font). I metodi di questa classe sono:

getHeight()	fornisce l'altezza del font (somma di *ascent*, *descent* e *leading*)
charWidth(char)	fornisce la larghezza di un determinato carattere
stringWidth(String)	fornisce la larghezza totale di una determinata stringa
getAscent()	fornisce la distanza tra la linea base dei caratteri e l'estremità superiore
getDescent()	fornisce la distanza tra la linea base dei caratteri e l'estremità inferiore
getLeading()	fornisce la distanza tra *ascent* di una riga e *descent* di un'altra

Ecco un esempio di codice per sistemare una stringa verticalmente e orizzontalmente al centro di un rettangolo.

(v pag.167)#

```
g.getFontMetrics().getHeight();  // altezza in pixel dei caratteri
g.getFontMetrics().charWidth(c); // larghezza in pixel del carattere c
g.getFontMetrics().stringWidth(s); //larghezza in pixel di una stringa
```

Font disponibili su un sistema

Per vedere quale sono i font disponibili su un sistema si può usare il metodo **getFontList()** di `Toolkit.getDefaultToolkit()` (il Toolkit dell'AWT è usato per legare le classi astratte AWT ad un particolare implementazione nativa del Toolkit, il suo metodo `getDefaultToolkit()` restituisce il Toolkit di default e `getFontList()` restituisce i nomi dei font disponibili sotto forma di un array di stringhe), come è fatto nell'esempio seguente:

```
// I09listaFont.java (F.Spagna) Lista dei font disponibili

import java.awt.Toolkit;

public class I09listaFont {

   public static void main(String args[]) {
      String font[] = Toolkit.getDefaultToolkit().getFontList();
      System.out.println("Disponibili i " + font.length + " font seguenti:");
      for (int n = 0; n < font.length; n++)
         System.out.println("   - " + font[n]);
      System.exit(0);
   }
}
```

che dà ad esempio sul nostro sistema il risultato:

```
Disponibili i 9 font seguenti:
   - Dialog
   - SansSerif
   - Serif
   - Monospaced
   - Helvetica
   - TimesRoman
   - Courier
   - DialogInput
   - ZapfDingbats
```

2.1.13 Grafica animata

2.1.13.1 Necessità di un thread per l'animazione

Immaginiamo di voler programmare un'animazione consistente in un rettangolino nero che si muove entro l'area di un'applet procedendo da sinistra a destra un pixel alla volta (ogni secondo). Se si fa un loop dentro il metodo `init()`, che è il primo dell'applet a partire, facendogli chiamare ciclicamente con un `while` il `repaint()` con il codice riportato qui di seguito, si ha la sorpresa di trovare che l'animazione non funziona e ciò è dovuto al fatto che il `while` monopolizza tutta l'attività del processore tanto da non permettere neanche il disegno da parte del `paint()`.

```
// I10motoNo.java (F.Spagna) Animazione non funzionante senza un thread

import java.awt.*;

public class I10conta extends java.applet.Applet {

   int i;
   public void init() {
      while (true) {
            try {
         Thread.sleep(1000);
            } catch(InterruptedException e) {}
         repaint();
         i++;
      }
   }
   public void paint(Graphics g) {
      g.fillRect(5*i % 100, 10, 5, 10);
   }
}
```

La soluzione per ottenere l'animazione sta nel fare agire l'applet in un thread che permetta la coesistenza delle sue azioni con le altre operazioni del sistema. Ecco allora il codice che in questo modo fa funzionare correttamente l'applet:

```
// I11moto.java F.Spagna Animazione funzionante con un thread

import java.awt.*;

public class I11conta extends java.applet.Applet implements Runnable {
   int i;
   Thread th;
   public void start() {
      if (th == null) {
         th = new Thread(this);      // si istanzia un thread per l'applicazione
         th.start();
      }
   }
   public void stop() {
      if (th != null) {
         th.stop();
         th = null;
      }
   }
   public void run() {
      while (true) {
            try {
         Thread.sleep(1000);
            } catch(InterruptedException e) {}
         repaint();
         i++;
      }
   }
   public void paint(Graphics g) {
      g.fillRect(5*i % 100, 10, 5, 10);
```

```
    }
}
```

Un'applet che lavori in un thread deve implementare l'interfaccia `Runnable`, deve avere una variabile d'istanza di tipo `Thread` che rappresenta (referenzia) il thread di esecuzione dell'applet, un metodo `start()` nel quale si fa partire il thread, uno `stop()` che lo ferma ed un metodo `run()` in cui si definisce tutta l'attività dell'applet. La classe `Thread` appartiene al package `java.lang`, che non è necessario importare in quanto esso è importato automaticamente in ogni programma Java. L'interfaccia `Runnable` presenta il solo metodo `run()` che parte subito dopo il metodo `start()` dell'applet se in questo si fa partire un thread. Il metodo `run()` rappresenta tutto il funzionamento del thread in quanto contiene il codice delle operazioni che si vogliono fare svolgere in modo indipendente nel thread (per esempio un loop per un'animazione). I metodi che vengono chiamati all'interno di `run()` lavorano anch'essi nel thread.

E' da tenere presente che, soprattutto nelle animazioni, non sempre un `repaint()` trova immediatamente il `paint()` conseguente disponibile, ma in certe condizioni, soprattutto se le risorse di sistema sono troppo sfruttate, il `paint()` richiesto da un `repaint()` può non essere eseguito subito e, se le richieste continuano, può formarsi una coda di richieste di `paint()` e in queste condizioni Java può anche decidere di eseguire solo l'ultima chiamata, saltando le precedenti in attesa. Un effetto di questo tipo si nota anche nell'esecuzione dell'applet precedente.

2.1.13.2 Metodi `paint()`, `update()`, `repaint()` e animazioni

Il metodo `repaint()` della classe `Applet` serve per ridisegnare l'area dell'applet, con il metodo `paint()`, che esso richiama, ma, diversamente da `paint()`, che viene mandato in esecuzione automaticamente dal browser quando ce n'è bisogno, esso deve essere richiamato esplicitamente nel programma (`paint()` forzato a programma).

Prima di ridisegnare l'area dell'applet con un `paint()`, il metodo `repaint()` cancella ogni volta tutta quella preesistente (e ciò è dovuto al fatto che `repaint()` chiama `update()` il quale pulisce l'area del componente con il suo colore di fondo (cioè lo cancella) e quindi chiama il `paint()`, che disegna secondo quanto è definito in esso. Infatti se si va a vedere la definizione del metodo `update()` della classe `Component`:

```
public void update(Graphics g) {
   g.setColor(getBackground());
   g.fillRect(0, 0, width, height);
   g.setColor(getForeground());
   paint(g);
}
```

si vede come, prima di chiamare il metodo `paint()`, esso cancella l'area.

Tutto ciò, nel caso di esecuzioni successive del metodo ripetute in un loop (per esempio se il `repaint()` è in un `while` per produrre una grafica animata mediante una successione di operazioni grafiche), produce un effetto di immagine intermittente (in inglese *flickering* o *blinking*).

Le soluzioni per ridurre questo effetto possono essere:

- ridefinire il metodo `update()` lasciandogli il solo `paint()`,
- ridurre al minimo necessario l'area che viene ridisegnata ogni volta,
- preparare prima il disegno in memoria (*double buffering*).

Vedremo in dettaglio nei tre paragrafi seguenti i tre accorgimenti.

2.1.13.3 Ridefinizione di `update()`

Per evitare la cancellazione dello schermo ad ogni invocazione del metodo `paint()` ogni volta che viene ridisegnata l'applet si può ricorrere alla ridefinizione del metodo `update()` dell'applet: questo metodo, che come si è detto nel paragrafo precedente, è chiamato automaticamente ogni volta che l'area dell'applet viene ridisegnata con un `repaint()`, prima di chiamare il `paint()` cancella l'area dell'applet: se allora si ridefinisce `update()` facendogli fare la sola operazione di richiamare il `paint()`:

```
public void update(Graphics g) {
   paint(g);
}
```

la cancellazione dell'area insita nella versione della superclasse del metodo non ha luogo e questo riduce l'effetto di *flickering*.

2.1.13.4 Riduzione dell'area ridisegnata

Oltre che con il sistema del paragrafo precedente, dal punto di vista della fluidità dell'animazione le cose possono essere migliorate anche riducendo l'area ridisegnata ogni volta (cioè quella soggetta a *flicker*) alle dimensioni strettamente necessarie con il metodo `clipRect(x1, y1, larghezza, altezza)`.

Ritornando alla ridefinizione di `update()` ricordata al paragrafo precedente, le cose migliorerebbero scrivendo:

```
public void update(Graphics g) {

   g.clipRect(x1, y1, larghezza, altezza);
   paint(g);
}
```

2.1.13.5 Tecnica del *double buffering* delle immagini

Ma con gli accorgimenti precedenti i problemi del *flickering* sono solo in parte risolti: per migliorare decisamente la qualità dell'animazione si può ancor meglio ricorrere alla tecnica del *double buffering*.

Il *double buffering* (in italiano potrebbe essere chiamato "doppio tampone", ma è il termine inglese che viene comunemente usato) è una tecnica che può essere adoperata con vantaggio

quando si deve disegnare una sequenza di immagini sull'area di un'applet e più in generale di un `Panel` o di un `Frame`, e che consiste nel preparare di volta in volta il disegno, compreso l'inserimento di eventuali immagini, non sull'area stessa da disegnare, ma fuori schermo in un oggetto (contesto grafico) di tipo `Graphics` creato in memoria (buffer) (operazione che Java permette di fare agevolmente) e poi riprodurlo tutto insieme in un solo colpo e quindi velocemente copiando l'area della memoria sull'area da disegnare (per esempio quella di un'applet) e sovrapponendola a quella preesistente. Tale tecnica permette di ottenere una successione di immagini più continua e fluida, senza il tremolio tipico prodotto da un disegno diretto delle immagini sullo schermo.

Il disegno viene preparato su un oggetto di tipo `Image`, creato con il metodo `createImage()` della superclasse `Component` che crea un'immagine disegnabile fuori schermo e a questo oggetto è associato un contesto grafico di tipo `Graphics`, che è creato mediante il metodo `getGraphics()` dell'oggetto di tipo `Image` che serve per disegnare sull'immagine stessa fuori schermo.

Ogni volta che, richiamato dal metodo `repaint()` inserito in un loop (che è posto il più delle volte nel metodo `run()` dell'applet implementata come `Runnable`), viene eseguito il `paint()`, esso può disegnare subito con un `drawImage()` l'immagine che era stata già preparata in memoria (operazione che in tal caso è molto veloce) e l'applicazione diventa disponibile per riprendere di nuovo le operazioni di disegno in memoria preparandosi alla prossima invocazione del metodo `paint()`, mentre l'immagine sullo schermo non viene toccata e rimane stabile. Notiamo che l'ultimo argomento di `drawImage()` prevede il parametro che specifica il richiedente del ...# e nel nostro caso questo è rappresentato dall'applet stessa, e quindi al metodo viene passato il `this`, che è la referenza dell'oggetto (l'applet) su cui è presentata la sequenza di immagini.

Questa tecnica è la più efficace perchè elimina radicalmente il *flickering*, però ha lo svantaggio di essere meno efficiente delle altre in termini di prestazioni ed esigente in fatto di memoria e perciò è meglio non usarla quando può essere evitata.

Quando si programma il metodo `paint()`, per ragioni di efficienza, soprattutto nel caso di animazioni, si deve evitare per quanto possibile di caricarlo con delle operazioni che potrebbero essere fatte altrove, in quanto quando il metodo è chiamato è bene faccia tempestivamente le cose che è chiamato a fare, e cioè il disegno.

Segue un programma di esempio con un'animazione costituita da una scritta che scorre sullo schermo insieme ad un'immagine (un piccolo aereo).

```
// I12doubuf045.java (F.Spagna) Scritta scorrevole con il double buffering

import java.awt.*;

public class I12doubuf045 extends java.applet.Applet
                       implements Runnable {
   Image aereo;
   Image imgMemo;
   Graphics gr;
   Thread th;
   String scrit;
   int bas, alt;
   Color colScrit, colFondo;
   int i;
```

```
    public void init() {
        scrit = getParameter("scritta");
        bas = Integer.parseInt(getParameter("base"));
        alt = Integer.parseInt(getParameter("altezza"));
        colScrit = new Color(Integer.parseInt(getParameter("colorscr"), 16));
        colFondo = new Color(Integer.parseInt(getParameter("colorfond"), 16));
        aereo = getImage(getCodeBase(), "aereo.gif");
        imgMemo = createImage(bas, alt);
        gr = imgMemo.getGraphics();
    }
    public void start() {
        if (th == null) {
            th = new Thread(this);
            th.start();
        }
    }
    public void stop() {
        if (th != null) {
            th.stop();
            th = null;
        }
    }
    public void run() {
        while (true) {         try {
            Thread.sleep(100);  } catch(InterruptedException e){}
            preparaImgMemo();
            repaint();
            i++;
        }
    }
    void preparaImgMemo() {
        gr.setColor(colFondo);
        gr.fillRect(0, 0, bas, alt);
        gr.setColor(colScrit);
        gr.setFont(new Font("TimesRoman", Font.BOLD, 28));
        gr.drawString(scrit, i % bas, alt - 20);
        gr.drawString(scrit, i % bas - bas, alt - 20);
        gr.drawImage(aereo, i % bas - 50, 5, 40, 28, this);
    }
    public void update(Graphics g) {
        paint(g);
    }
    public void paint(Graphics g) {
        g.drawImage(imgMemo, 0, 0, this);
    }
}
```

Si noti in particolare nel codice:

- la definizione delle due variabili d'istanza di tipo `Image` e `Graphics` rispettivamente per l'immagine da preparare in memoria e per il contesto grafico associato ad essa, variabili istanziate nel metodo `init()`;

- la creazione di un metodo speciale (cui abbiamo dato il nome `preparaImgMemo()`) per preparare l'immagine fuori schermo in memoria che è chiamato ad ogni ciclo dal `while` entro il metodo `run()` e che raccoglie tutte le operazioni grafiche fatte in memoria;

- la semplicità assoluta del metodo `paint()` cui viene affidata la sola funzione di riprodurre in un colpo solo sullo schermo l'immagine precedentemente preparata in memoria: infatti è bene sempre per ragioni di efficienza evitare di fare in questo metodo operazioni che possono invece essere fatte fuori, soprattutto per le animazioni in quanto, quando il metodo è chiamato, deve essere pronto per disegnare subito;

- la solita ridefinizione del metodo `update()` per eliminare la cancellazione dello schermo ad ogni `repaint()`.

Al programma dimostrativo, di per sé molto semplice, abbiamo aggiunto alcune altre cose:

- per generalizzare l'applet e renderla flessibile nell'uso abbiamo aggiunto la possibilità di inserire alcuni dati come parametri passati ad essa dall'HTML, come il contenuto della stringa, la base e l'altezza entro cui scorre la scritta, il colore della scritta ed il colore di fondo;

- il contenuto della scritta viene passato come stringa e come tale utilizzato all'interno dell'applet e quindi non richiede alcun trattamento;

- i parametri base e altezza sono passati come stringhe, ma sono tradotti nel programma in interi con la funzione `Integer.parseInt(String);`

- i due parametri dei colori sono passati come stringhe che esprimono il colore in valore esadecimale e devono essere poi tradotti in termini di colore nel programma prima calcolandone il valore intero con il metodo `Integer.parseInt(String, 16)` questa volta con una radice 16 (relativa ai numeri esadecimali) e poi il colore viene creato con un construttore avente un argomento così rappresentato;

- per riprendere da sinistra la scritta scorrevole via via che essa scompare sulla destra, se ne scrive sempre una seconda sfalsata all'indietro di un numero di pixel uguali a quello della lunghezza della base dell'area dell'applet, e questo funziona bene anche quando l'ascissa di inizio della stringa è negativo, cioè fuori dell'area dell'applet.

Si riproduce in figura 9.23 la figura ripresa al volo in certo istante per un'applet lanciata da un HTML così scritto:

```
<html>
<applet code=doubuf045.class width=400 height=40>
   <param name=scritta value="questo è un aereo che vola">
   <param name=base value="400">
   <param name=altezza value="50">
   <param name=colorscr value="0000ff">
   <param name=colorfond value="ffff00">
</applet>
```

n aereo che vola ✈ **questo è u**

Figura 9.23 Animazione con una scritta scorrevole.

2.1.13.6 Animazioni con immagini

Un'animazione con delle immagini può essere fatta in un'applet che lavora in un thread presentando in successione una serie di immagini una di seguito all'altra come in un film in una posizione fissa sullo schermo, e in tal caso l'effetto di movimento è prodotto soltanto dal contenuto delle immagini, o anche, se si vuole, con uno spostamento della posizione dell'immagine, con intervalli di tempo abbastanza piccoli da produrre la sensazione dell'animazione.

La serie di immagini, che di solito vengono poste tutte in una sottodirectory di quella dove è situata l'applet e cioè il suo bytecode, sono in genere rappresentate nel programma da un array (o ancor meglio un `Hashtable`) di oggetti di tipo `Image`. Se per esempio supponiamo che le immagini sono in numero di 10 e i file relativi si chiamano img0.gif, img1.gif, ... img9.gif e sono poste in una sottodirectory `immagini/` di quella dell'applet, si scriverebbe:

```
Image img[] new Image[10]                    // variabile d'istanza della classe
for (int n = 0; n < img.length; n++)                    // nel metodo init()
   img[n] = getImage(getCodeBase(), "immagini/img" + n + ".gif");
```

(si noti il modo adoperato per chiamare file con nomi diversi con un unico loop).

Un esempio interessante di animazione in Java è rappresentato dall'applet Animator che si trova negli esempi del JDK.

2.1.13.7 Caricamento delle immagini e classe `MediaTracker`

La classe **`MediaTracker`** permette di creare degli oggetti che sono capaci di sorvegliare (*track*) lo stato di vari oggetti multimediali, anche se per il momento sono supportate soltanto le immagini. Per far questo si crea un'istanza della classe e si aggiungono ad essa le immagini di cui si vuol seguire lo stato con il metodo `addImage()`. A ciascuna immagine si può assegnare un numero di identificazione che ne indicherà la posizione in sequenza, potendo una serie di immagini avere lo stesso numero e quindi essere trattata nel suo insieme.

Variabili di classe pubbliche:

ABORTED	Flag indicante che il caricamento non è riuscito
COMPLETE	Flag indicante che il caricamento è stato completato con successo
ERRORED	Flag indicante che il caricamento ha incontrato un errore
LOADING	Flag indicante che il caricamento è in corso

Costruttore:

`MediaTracker(Component)`
Creates a Media tracker to track images for a given Component.

Metodi:

addImage(Image, int)
aggiunge un'immagine alla lista di immagini da sorvegliare

addImage(Image, int, int, int)
aggiunge un'immagine ridimensionata alla lista di immagini da sorvegliare

checkAll()
vede se tutte le immagini hanno finito di essere caricate, ma non inizia il caricamento se questo non è già in atto

checkAll(boolean)
vede se tutte le immagini hanno finito di essere caricate

checkID(int)
vede se tutte le immagini marcate con un dato ID hanno finito di essere caricate, ma non inizia il caricamento se questo non è già in atto

checkID(int, boolean)
vede se tutte le immagini marcate con un dato ID hanno finito di essere caricate

getErrorsAny()
restituisce una lista di tutti i media che hanno incontrato un errore

getErrorsID(int)
restituisce una lista di media con un dato ID che hanno incountrato un errore

isErrorAny()
vede lo stato di errore di tutte le immagini

isErrorID(int)
vede lo stato di errore di tutte le immagini con un dato ID

removeImage(Image)
rimuove dal MediaTracker l'Image specificata

removeImage(Image, int)
rimuove dal MediaTracker l'Image specificata con un dato ID

removeImage(Image, int, int, int)
rimuove dal MediaTracker l'Image specificata con una data dimensione e ID

statusAll(boolean)
restituisce il booleano OR dello stato di tutti i media da sorvegliare

statusID(int, boolean)
restituisce il booleano OR dello stato di tutti i media con un dato ID

waitForAll()
inizia il caricameto di tutte le immagini

waitForAll(long)
inizia inizia il caricameto di tutte le immagini

waitForID(int)
inizia il caricameto di tutte le immagini con un dato ID e aspetta finchè hanno inito di

essere caricate o ricevono un errore

waitForID(int, long)

inizia il caricameto di tutte le immagini con un dato ID

Qui viene fatto un esempio che prevede un'immagine di sfondo che viene riprodotta per successivi incrementi via via che viene caricata ed una serie di immagini sovrapposte a quella di un'animazione di cui si aspetta il caricamento completo prima di presentarle sullo schermo per evitare di avere un'animazione parziale, che non sarebbe molto bella.

```
// I13anim058.java (F.Spagna) Animazione con immagini
// 01-16.11.98 (inizio 14 novembre 1998)

import java.awt.*;

public class I13anim058 extends java.applet.Applet implements Runnable {
   final int NIMG = 18;
   Image img[] = new Image[NIMG];   // numero di immagini usate nell'animazione
   Image sfondo;                    // array delle immagini istanziato
   int im;                          // immagine di sfondo
   boolean caricate;                // indice dell'immagine corrente da disegnare
   MediaTracker tracker;            // indica se tutte le immagini sono state caricate
   Thread th;                       // tracker
                                    // oggetto Thread dell'applet
  public void init() {
     tracker = new MediaTracker(this);
     sfondo = getImage(getDocumentBase(), "images/background.gif");  // istanzia il tracker
     tracker.addImage(sfondo, 0);     // aggiunge immagine di fondo con id = 0
     for (int i = 1; i <= NIMG; i++) {          // ogni immagine dell'animazione
        img[i-1] = getImage(getDocumentBase(),         // viene stabilita
                      "images/img00" + ((i<10) ? "0" : "") + i + ".gif");
        tracker.addImage(img[i-1], 1);      // e aggiunta al tracker con id = 1
     }
  }
 public void paint(Graphics g) {
    g.drawString("Caricamento immagini...", 10, 20); //messaggio d'attesa
    if ((tracker.statusAll(false) & MediaTracker.ERRORED) != 0) {
       g.setColor(Color.red); //se errori caricamento immagini rettang.rosso
       g.fillRect(0, 0, size().width, size().height);
       return;
    }
    g.drawImage(sfondo, 0, 30, this);//sempre disegna sfondo incrementalmente
    if (tracker.statusID(1, false) == MediaTracker.COMPLETE)
       g.drawImage(img[im], 100, 100, this);//disegna solo se caricate tutte
 }
public void run() {
                 try {                    // prima di cominciare l'animazione
   tracker.waitForID(0); //aspetta che immagine di fondo sia tutta caricata
   tracker.waitForID(1);//aspetta che immagini di animazione siano caricate
   // tracker.waitForAll();     // aspetta finche'tutte le immagini caricate
                 } catch (InterruptedException e) { return; }
   im = 0;
   while (true) {                          // comincia con la prima immagine
      repaint();                                  // cicla indefinitamente
      try { Thread.sleep(150); } catch(InterruptedException e) { break; }  // disegna l'immagine corrente
```

```
            im++;
            im %= NIMG;                          // incrementa l'indice dell'immagine
        }
    }
    public void update(Graphics g) {
        paint(g);                                       // per eliminare il tremolio
    }
    public void start() {
        th = new Thread(this);               // fa partire il thread dell'animazione
        th.start();
    }
    public void stop() {
        th.stop();                                 // ferma il thread dell'animazione
        th = null;
    }
}
```

```
try {
      tracker.waitForAll();          // aspetta finche'tutte le immagini caricate
      caricate = !tracker.isErrorAny();       // il tracker informa se caricate
/*                   } catch (InterruptedException e) { }
//-----------------------------------------------------------------------------

class fram extends Frame {                       // finestra che si puo' chiudere

   public fram(String str) {
      super (str);
   }
//-----------------------------------------------------------------------------

   public boolean handleEvent(Event evt) {
      switch (evt.id) {
         case Event.WINDOW_DESTROY:
            dispose();
            System.exit(0);
            return true;
         default:
            return super.handleEvent(evt);
      }
   }
//-----------------------------------------------------------------------------
}
*/
```

The init() method is called by the AWT when an applet is first loaded or reloaded. Override this method to perform whatever initialization your applet needs, such as initializing data structures, loading images or fonts, creating frame windows, setting the layout manager, or adding UI components.

destroy() Place additional applet clean up code here. destroy() is called when your applet is terminating and being unloaded.

The start() method is called when the page containing the applet first appears on the screen. The AppletWizard's initial implementation of this method starts execution of the applet's thread.

The stop() method is called when the page containing the applet is no longer on the screen. The AppletWizard's initial implementation of this method stops execution of the applet's thread.

The run() method is called when the applet's thread is started. If your applet performs any ongoing activities without waiting for user input, the code for implementing that behavior typically

goes here. For example, for an applet that performs animation, the run() method controls the display of images.

meaning that another thread has interrupted this one

This frame class acts as a top-level window in which the applet appears when it's run as a standalone application.

The handleEvent() method receives all events generated within the frame window. You can use this method to respond to window events. To respond to events generated by menus, buttons, etc. Or other controls in the frame window but not managed by the applet, override the window's action() method.

For each image in the animation, this method first constructs a string containing the path to the image file; then it begins loading the image into the img array. Note that the call to getImage will return before the image is completely loaded.

If re-entering the page, then the images have already been loaded. caricate == TRUE.

```
Rectangle r = g.getClipRect();
g.clearRect(r.x, r.y, r.width, r.height);
```

update paint() The background image fills our frame so we don't need to clear the applet on repaints, just call the paint method.

run() Run the animation thread.

```
    if ((tracker.statusAll(false) & MediaTracker.ERRORED) != 0) {
// Paint a large red rectangle if there are any errors loading the images
        g.setColor(Color.red);
        g.fillRect(0, 0, size().width, size().height);
        return;
    }
    g.drawImage(sfondo, 0, 0, this);
```

Otherwise always paint the background so that it appears incrementally as it is loading.

if (tracker.statusID(1, false) == MediaTracker.COMPLETE)

Finally, only paint the current animation frame if all of the frames (id == 1) are done loading so that we don't get partial animations.

```
/*
            if (standAlone)
               img[i-1] = Toolkit.getDefaultToolkit().getImage(strImg);
            else
               img[i-1] = getImage(getDocumentBase(), strImg);
*/
/*
   boolean   standAlone = false;                   // posto true se standalone

   public static void main(String args[])    {

      fram fr = new fram("Animazione");
      fr.show();       // necessario perche' insets() restituisca valori validi
      fr.hide();
      fr.resize(fr.insets().left + fr.insets().right  + 320,
               fr.insets().top   + fr.insets().bottom + 240);
      anim060 a60 = new anim060();
      fr.add("Center", a60);
```

```
        a60.standAlone = true;
        a60.init();
        a60.start();
        fr.show();
    }
*/
```

2.2 Grafica a 2 dimensioni

2.2.1 L'API 2D di Java

Nella versione 1.2 del JDK sono state introdotte funzionalità di grafica a due dimensioni con l'API Java 2D facente parte del core stesso dell'AWT che permettono una grafica avanzata con il disegno di forme geometriche complesse e la trasformazione del sistema di coordinate, e mettono a disposizione mezzi più sofisticati di quelli precedenti per il disegno di testi e font, il trattamento delle immagini, la gestione dei colori e la stampa indipendente dal dispositivo (*device-independent*).

Mentre nel JDK 1.1 si potevano disegnare solo alcune forme semplici (rettangoli, ellissi, poligoni) con una linea di spessore fissato, con questa nuova libreria si può disegnare qualunque tipo di forma con diversi stili di linea e di riempimento. Gli oggetti grafici disegnati possono poi essere trasformati con una traslazione, rotazione, cambiamento di scala o scorrimento (*shear*). I colori possono essere sovrapposti con un effetto di trasparenza.

[8.1] Bill Loeb, The Java 2D API, Dr.Dobb's Journal, #296, February 1999, page 44.

2.2.2 Classe `Graphics2D`

Alla base di tutta la grafica bidimesionale sta la classe astratta **`Graphics2D`** del package `java.awt` che estende la classe `Graphics` dello stesso package. Generalmente nel metodo `paint()` dell'applet o dell'applicazione viene creato un oggetto di classe `Graphics2D` ottenuto facendo il casting dell'oggetto di tipo `Graphics` che è passato come argomento dal sistema al metodo:

```
Graphics2D g2D = (Graphics2D)g;
```

Una differenza importante con la grafica dell'AWT 1.0 è la possibilità di specificare le coordinate dei punti nel piano con numeri in virgola mobile anzichè interi.

Metodi:

`public abstract void draw(Shape s)`
disegna una forma

`public abstract void fill(Shape s)`
disegna una forma piena

`public abstract void drawImage(diverse signature)`
disegna un'immagine

`public abstract void drawRenderedImage(diverse signature)`
disegna un'immagine applicando la trasformazione corrente

public abstract void drawString(diverse signature)
disegna una stringa

public abstract void hit(argomenti)
va a vedere se

public abstract void hitString(argomenti)
va a vedere se

public abstract GraphicsConfiguration getDeviceConfiguration()
restituisce la device configuration

public abstract void setComposite(argomenti)
va a vedere se

public abstract void setPaint(argomenti)
va a vedere se

public abstract void setStroke(argomenti)
va a vedere se

public abstract void setRenderingHints(argomenti)
va a vedere se

public abstract int getRenderingHints(argomenti)
restituisce

public abstract void translate(vari argomenti)
v

public abstract void rotate(vari argomenti)
v

public abstract void scale(argomenti)
v

public abstract void shear(argomenti)
v

public abstract void hit(argomenti)
v

public abstract void transform(AffineTransform)
trasforma gli assi dell'utilizzatore

public abstract void setTransform(AffineTransform)
t

public abstract AffineTransform getTransform()
restituisce

public abstract Paint getPaint()
restituisce

public abstract Composite getComposite()
restituisce

public abstract void setBackground(Color)
t

public abstract Color getBackground()
restituisce

```
public abstract Stroke getStroke()
        restituisce

public abstract void clip(Shape s)
        interseca il clip corrente con l'interno di una data forma
```

2.2.3 Linea passante per più punti

La classe **GeneralPath**, contenuta nel package `java.awt.geom`, rappresenta una linea che passa per più punti, come la `polygon` dell'AWT 1.0, ma con la possibilità di unire i punti non solo con segmenti di retta, ma anche con curve quadratiche o cubiche di tipo curve di Bezier, permettendo quindi la creazione di forme complesse. Un `GeneralPath` può contenere diversi subpaths.

Le curve tracciate tra le coppie di punti successivi possono essere del primo ordine o lineari (nessun punto di controllo intermedio), del secondo ordine o quadraticche (un punto di controllo intermedio) oppure del terzo ordine o cubiche (due punti di controllo intermedi) usando una curva di Bezier. Le curve di Bezier sono curve polinomiali.

La regola di avvolgimento (*winding rule*) specifica come viene determinato l'interno della figura delimitata dalla linea: la regola `EVEN_ODD` e quella `NON_ZERO` (per la spiegazione di questa caratteristica si rimanda a quella data nel JDK 1.2 stesso).

Questa classe implementa l'interfaccia `Shape`.

Metodi:

```
GeneralPath()
        diversi costruttori, tra cui uno che riceve un parametro che specifica la regola di
        winding ehe può essere stabilito mediante il valore della costante di classe
        GeneralPath.EVEN_ODD oppure GeneralPath.NON_ZERO
moveTo(float x, float y)
        aggiunge un punto alla linea spostandosi sulle sue coordinate senza tracciare alcuna linea
lineTo(float x, float y)
        aggiunge un punto alla linea tracciando un segmento di retta tra l'ultimo punto
        ed uno successivo (x, y)
quadTo(float x1, float y1, float x, float y)
        aggiunge due punti alla linea tracciando una curva quadratica tra l'ultimo
        punto ed uno successivo (x, y) con un punto intermedio di controllo (x1, y1)
curveTo(float x1,float y1, float x2,float y2, float x, float y)
        aggiunge tre punti alla linea tracciando una curva di Bezier cubica tra l'ultimo punto
        ed uno successivo (x, y) con due punti intermedi di controllo (x1, y1) e (x2, y2)
closePath()
        chiude la linea ritornando al punto imiziale con una linea retta
```

append(Shape s, boolean connect)
aggiunge alla linea un oggetto di tipo `Shape`

append(PathIterator pi, boolean connect)
aggiunge alla linea un oggetto di tipo `PathIterator`

transform(AffineTransform)
trasforma la geometria della linea secondo una trasformazione data

boolean contains(diversi possibili argomenti specificanti un punto)
va a vedere se un punto è contenuto nella figura racchiusa dalla linea

Rectangle getBounds()
restituisce il rettangolo delimitante la linea

Esempio:#

2.2.4 Traslazione e rotazione degli assi

La classe **AffineTransform**, contenuta nel package `java.awt.geom`, rappresenta una trasformazione affine con conservazione delle linee rette e del parallelismo tra rette che permette di effettuare la traslazione. la rotazione, il cambiamento di scala e lo shear degli assi di riferimento e quindi di una figura. variando il cosiddetto spazio dell'utilizzatore (*user space*) rispetto allo spazio del dispositivo (*device space*).

Metodi:

setToTranslation(float, float)
effettua la traslazione di un certo valore per x ed un altro valore per y

setToRotation(float)
effettua la rotazione di un certo numero di radianti

2.2.5 Font e testo 2D

Con l'API Java "D le stringhe possono essere disegnate con tradformazione affine come qualunque altra forma disegnata. E' anche possibile ottenere un oggetto di tipo `Shape` che rappresenti la stringa come disegno con il quale possono essere fatte tutte le operazioni possibili con qulasiasi altro elemento grafico.

Il testo dopo una rotazione risulta dentellato.

[8.2] Bill Day, Java Media, Getting started with Java 2D, Java World, July 1998.

2.3 Trattamento avanzato delle immagini

2.3.1 Creazione di un'immagine JPEG

Le classi del package `com.sun.image.codec.jpeg` permettono di codificare in formato JPEG e salvare in un file JPG un'immagine creata come oggetto della classe `Image`. Questo può essere particolarmente utile ad esempio nel caso di un serverlet che debba trasmettere al cliente con una pagina HTML un'immagine creata al volo in relazione alle circostanze della richiesta.

Per far questo si crea l'immagine come oggetto della classe `BufferedImage`, si disegna quello che si vuole sul suo contesto grafico (oggetto di tipo `Graphics`) e si codifica su un `BufferedOutputStream` l'immagine sottoposta al'encoder rappresentato da un'istanza di `JPEGImageEncoder`. L'esempio seguente mostra tutto ciò:

L'immagine così creata e salvata in un file `immag.jpg` è mostrata nella figura 9.24 seguente.

Figura 9.24 Immagine JPEG creata con encoder java.

2.4 Tavola riassuntiva del package `java.awt`

---

java.awt	**(Abstract Window Toolkit)**
	contiene le classi per la gestione di interfacce grafiche utente
class BorderLayout	layout riferito ai bordi
Button	bottone
Canvas	area di disegno
CardLayout	layout a strati sovrapposti
Checkbox	controllo di scelta si/no
CheckboxGroup	gruppo di `Checkbox`
CheckboxMenuItem	voce di un menù tipo `Checkbox`
Choice	menù a tendina
Color	colore
Dialog	finestra di dialogo
Dimension	larghezza e altezza di un'area rettangolare
Event	evento nell'interfaccia grafica
FileDialog	finestra di directory e file
FlowLayout	layout su righe orizzontali
Font	tipo di carattere delle scritte
FontMetrics	metrica di un tipo di carattere
Frame	finestra con titolo
Graphics	contesto grafico (contiene tutti i metodi grafici)
GridLayout	layout secondo una griglia
GridBagLayout	layout secondo una griglia evoluto
Image	immagine
Insets	distanze dai bordi
Label	etichetta con un testo
List	lista scorrevole
Menu	menù
Menubar	barra dei menù
MenuItem	voce di menù

Panel	pannello
Point	punto
Polygon	poligono
Rectangle	rettangolo
Scrollbar	barra di scorrimento
TextArea	testo editabile su più righe
TextField	testo editabile su una riga
Window	finestra base

---

java.awt.image contiene le classi per il trattamento delle immagini

---

java.awt.peer contiene le classi per

---

3. Java Foundation Classes

3.1 Le Java Foundation Classes (JFC)

Le **Java Foundation Classes** (JFC) sono

3.2 La libreria Swing

3.2.1 Le classi Swing

Le classi Swing

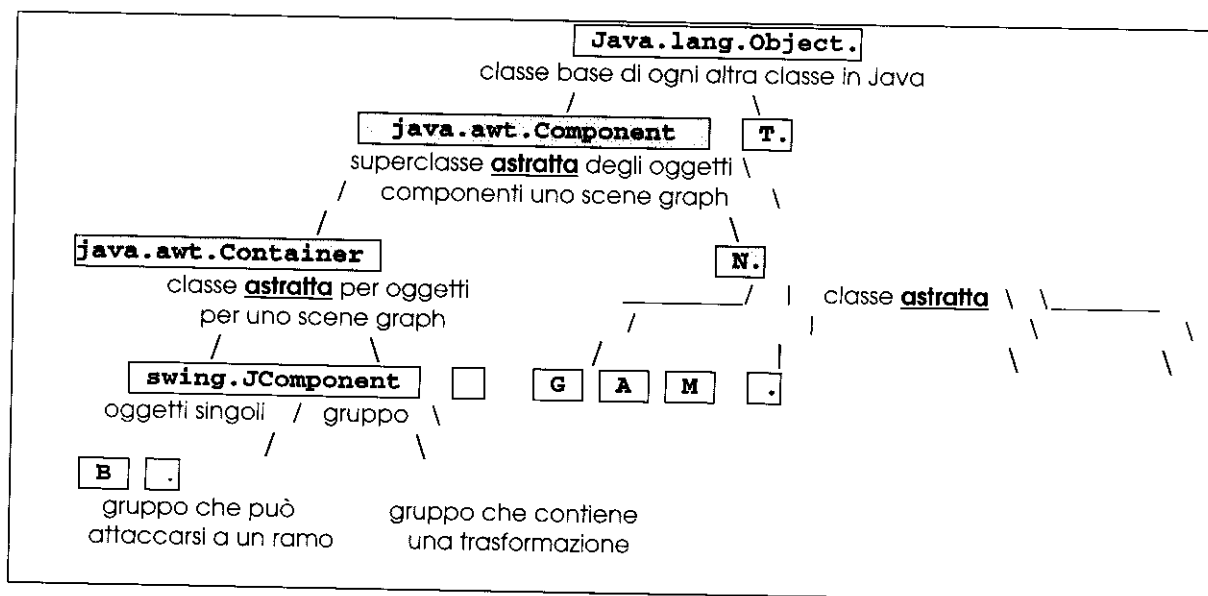



Figura 10.1 Schema di discendenza ereditaria delle classi Swing.

3.2.2 Classe `JFrame`

La classe `JFrame` rappresenta

E' qui riportato il codice di un semplice esempio di creazione di un frame tipo JFrame:

```
// J01frame.java (F.Spagna) Semplice esempio di JFrame

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class J01frame extends JFrame {
   Container cp;
   public frame(String titolo) {
      super(titolo);
      cp = this.getContentPane();
      cp.setBackground(Color.white);
      cp.setForeground(Color.blue);
      JLabel lab = new JLabel("Questo è un frame tipo Swing", JLabel.CENTER);
      lab.setFont(new Font("Sans", Font.BOLD, 20));
      cp.add(lab);
      addWindowListener(new WindowEventHandler());
      setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
      setSize(350, 200);
      show();
   }
   public static void main(String[] args) {
      new frame("Frame Swing");
   }
   class WindowEventHandler extends WindowAdapter {
      public void windowClosing(WindowEvent evt) {
         System.exit(0);
      }
   }
}
```

Il frame così creato può essere visto nella figura 10.2.

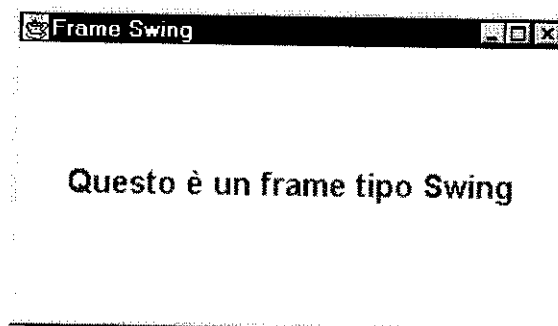
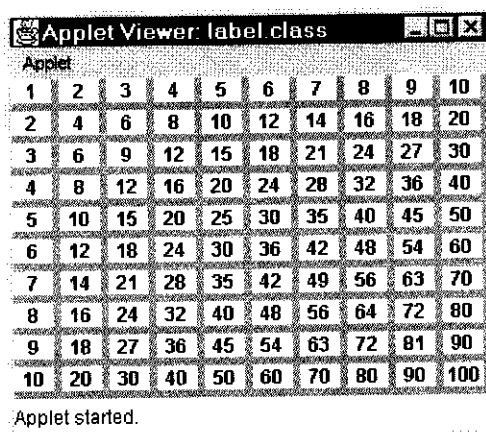



Figura 10.2 Esempio di finestra tipo `JFrame`.

3.2.3 Classe **JLabel**

La classe **JLabel** rappresenta

Un esempio di `JLabel`, che ripete quanto è già stato fatto con oggetti di tipo `Label` nell'esempio del paragrafo 7.2.2 (si trattava di disegnare una tavola pitagorica, vedi figura 7.3), è fatto qui di seguito:

```
// J02label.java (F.Spagna) Esempio elementare di JLabel

import java.awt.*;
import javax.swing.*;

public class J02label extends JApplet {

   public void init() {
      Container cp = getContentPane();
      cp.setLayout(new GridLayout(10, 10, 5, 5)); // 10 righe 10 colonne gap=5
      cp.setBackground(Color.green);
      for (int r = 1; r <= 10; r++)
         for (int c = 1; c <= 10; c++) {
            JLabel lab = new JLabel("" + r*c, JLabel.CENTER);
            lab.setHorizontalTextPosition(JLabel.LEFT);
            lab.setVerticalTextPosition(JLabel.TOP);
            lab.setBackground(Color.white);
            lab.setForeground(Color.black);
            lab.setOpaque(true);
            cp.add(lab);
         }
   }
}
```

Il risultato è visibile in figura 10.3.

Applet Viewer: label.class

Applet

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Applet started.

Figura 10.3 Esempio di JLabel.

3.2.3.1 Bordi dei JLabel

Le etichette di tipo JLabel possono essere dotate di bordi di vario aspetto. Nell'esempio seguente vengono provati un po' tutti i vari tipi disponibili:

```
// J03bordi.java (F.Spagna) Esempio di vari bordi sui JLabel

import javax.swing.border.*;
import javax.swing.*;
import java.awt.*;

public class J03bordi extends JApplet{

   static String titolo[] = { "Empty", "Etched", "Etched&Colored",
           "BevelSu", "BevelGiù", "BevelColor", "SoftBevel", "Matte",
           "Line", "Line", "Compound", "Titled" };

   public void init() {

      Icon icon = new ImageIcon("sun.gif");
      Border bordo[] = new Border[12];
      bordo[0] = new EmptyBorder(1, 1, 1, 1);                  // insets
      bordo[1] = new EtchedBorder(EtchedBorder.RAISED);
      bordo[2] = new EtchedBorder(EtchedBorder.LOWERED, Color.red,Color.blue);
      bordo[3] = new BevelBorder(BevelBorder.RAISED);
      bordo[4] = new BevelBorder(BevelBorder.LOWERED);
      bordo[5] = new BevelBorder(BevelBorder.RAISED, Color.gray,Color.yellow);
      bordo[6] = new SoftBevelBorder(BevelBorder.LOWERED);
      bordo[7] = new MatteBorder(20, 20, 20, 10, icon);
      bordo[8] = new LineBorder(Color.red, 5);           // spessore linea
      bordo[9] = LineBorder.createGrayLineBorder();
      bordo[10] = new CompoundBorder(new BevelBorder(BevelBorder.RAISED),
                                new EtchedBorder(EtchedBorder.RAISED));
      bordo[11] = new TitledBorder(new LineBorder(Color.red),
           "titolo", TitledBorder.DEFAULT_JUSTIFICATION,
           TitledBorder.CENTER, new Font("Sans", Font.BOLD, 16), Color.blue);

      JPanel pan = new JPanel(new GridLayout(3, 4, 5, 5));
      for (int n = 0; n < 12; n++) {
         JLabel lab = new JLabel(titolo[n], JLabel.CENTER);
         lab.setOpaque(true);
         lab.setBorder(bordo[n]);                 // assegna i bordi ai label
         pan.add(lab);
      }
      getContentPane().add(pan);
   }
}
```

In figura 10.4 si possono vedere su una griglia i vari tipi di bordi applicati alle diverse etichette del programma.

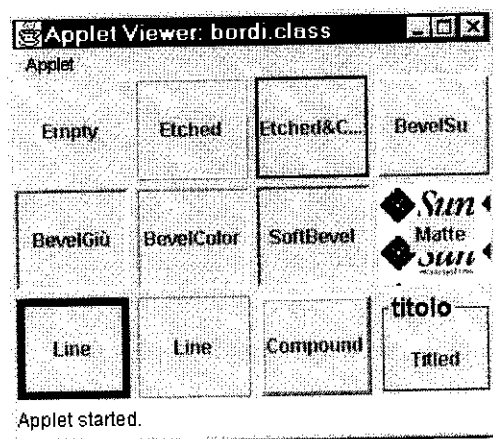



Figura 10.4 Vari tipi di bordi applicati ad etichette `JLabel`.

3.2.4 Classe **JTextField**

La classe **JTextField** rappresenta

Si fa qui un esempio di `JTextField` con un oggetto di questa classe che riceve i dati di input dell'utente ed un altro che mostra questi dati quando nel primo campo è battuto il tasto di invio.

```
// J04textfield.java (F.Spagna) Esempio di TJTextField

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class J04textfield extends JApplet {

    Container cp;
    JTextField tf1, tf2;
    JLabel lab1, lab2;
    public void init() {
        lab1 = new JLabel("entra dati:  ", JLabel.RIGHT);
        lab1.setFont(new Font("", Font.BOLD, 18));
        lab2 = new JLabel("mostra dati:  ", JLabel.RIGHT);
        lab2.setFont(new Font("", Font.BOLD, 18));
        cp = this.getContentPane();
        cp.setLayout(new GridLayout(2, 2, 0, 60));
        tf1 = new JTextField("",  20);
        tf1.setBorder(BorderFactory.createLineBorder(Color.black, 3));
        tf1.addActionListener(new TextFieldListener());
        tf2 = new JTextField(20);
        tf2.setBorder(BorderFactory.createLineBorder(Color.blue, 3));
        cp.add(lab1);
```

```
        cp.add(tf1);
        cp.add(lab2);
        cp.add(tf2);
    }
    class TextFieldListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
             tf2.setText(e.getActionCommand());
        }
    }
}
```

In figura 10.5 si vede l'applet dopo l'inserimento di un testo da parte dell'utente.

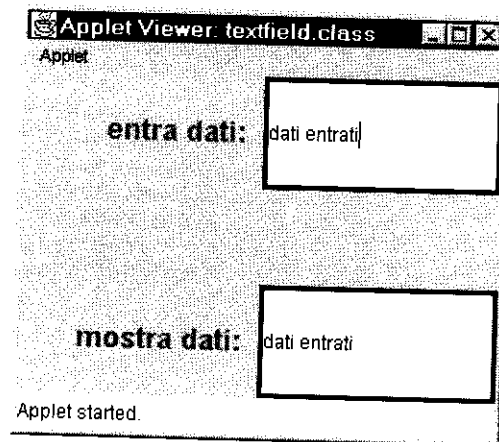



Figura 10.5 Esempio di due `JTextField` collegati.

3.2.5 Classe `JRadioButton`

La classe **`JRadioButton`** rappresenta

E' fatto qui di seguito un esempio di una serie di `JRadioButton` riuniti in un `JRadioButton` èpresenti su un'applet:

```
// J05radiob.java (F.Spagna) Esempio di JRadioButton riuniti in un ButtonGroup

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class J05radiob extends JApplet {
   JRadioButton rBot[] = new JRadioButton[3];
```

```
    JLabel lab;
    Container cp;

    public void init() {
       cp = getContentPane();
       cp.setLayout(new GridLayout(2, 1));
       ascolt asc = new ascolt();
       ButtonGroup grup = new ButtonGroup();
       JPanel pan = new JPanel();
       pan.setLayout(new GridLayout(5, 1));
       pan.add(new JLabel("Fai una scelta"));
       for (int n = 0; n < 3; n++) {
           rBot[n] = new JRadioButton("Scelta " + n, true);
           rBot[n].addActionListener(asc);
           grup.add(rBot[n]);
           pan.add(rBot[n]);
       }
       lab = new JLabel("Scelta 0", JLabel.CENTER);
       lab.setFont(new Font("Sans", Font.BOLD, 20));
       pan.add(lab);
       cp.add(pan);
    }
    class ascolt implements ActionListener {
       public void actionPerformed(ActionEvent e) {
           JRadioButton rBscelto = (JRadioButton) e.getSource();
           for (int n = 0; n < 3; n++)
              if (rBscelto == rBot[n]) {
                  lab.setText("Scelta " + n);
                  cp.validate();return;
              }
       }
    }
}
```

L'applet è presentata in figura 10.6.

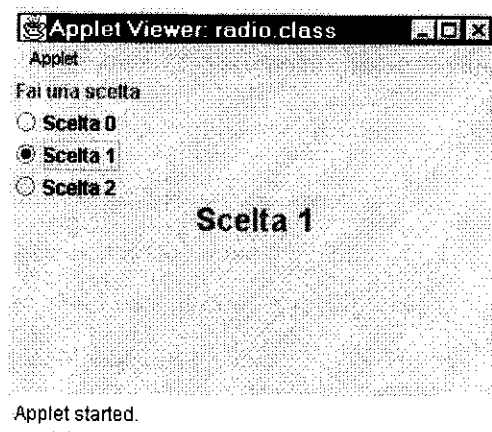



Figura 10.6 Esempio di `JRadioButton`.

3.2.6 Class **JMenu**, **JMenuBar** e **JMenuItem**

La classe **JMenu** rappresenta un menù provvisto di una serie di voci (item) costituite da oggetti della classe **JMenuItem** e disposto su una barra dei menù definita come istanza della classe **JMenu**.

La gerarchia delle suddette tre classi è la seguente:

```
java.lang.Object
 |_ java.awt.Component
       |_ java.awt.Container
             |_ java.swing.JComponent
                   |_ java.swing.JMenuBar
                         |_ java.swing.AbstractButton
                            |_ java.swing.JMenuItem
                                  |_ java.swing.JMenu
```

Un esempio di applicazione di queste tre classi è fatto del programma seguente:

```
// J06menu.java () Esempio di JMenu con JMenuBar e JMenuItem

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;

public class J06menu extends JApplet {

   JMenuItem itemBianco,  itemNero,  itemRosso, itemVerde, itemQuadrato,
             itemCerchio, itemPenta, itemEsa,   itemDodeca;
   Label lab;

   public void init() {
      Container container = this.getContentPane();

      container.setLayout(new BorderLayout());
      lab = new Label("*******");
      lab.setFont(new Font("", Font.BOLD, 16));
      container.add("South", lab);
      JMenuBar menuBar = new JMenuBar();
      menuBar.setBorder(new BevelBorder(BevelBorder.RAISED));
      menuBar.setBorderPainted(true);
      container.add(menuBar, BorderLayout.NORTH);

      JMenu menuColore = new JMenu("Colore", true);
      menuBar.add(menuColore);
      menuColore.add(itemBianco = new JMenuItem("Bianco"));
      menuColore.add(itemNero = new JMenuItem("Nero"));
```

```
        menuColore.addSeparator();
        menuColore.add(itemRosso = new JMenuItem("Rosso"));
        menuColore.add(itemVerde = new JMenuItem("Verde"));

        JMenu menuForma = new JMenu("Forma");
        menuBar.add(menuForma);
        JMenu menuPoli = new JMenu("Poligono");
        menuForma.add(menuPoli);
        menuForma.addSeparator();
        menuForma.add(itemQuadrato = new JMenuItem("Quadrato"));
        menuForma.add(itemCerchio = new JMenuItem("Cerchio"));

        menuPoli.add(itemPenta = new JMenuItem("Pentagono"));
        menuPoli.add(itemEsa = new JMenuItem("Esagono"));
        menuPoli.add(itemDodeca = new JMenuItem("Dodecagono"));

        ascoltaMenu am = new ascoltaMenu();

        itemBianco.addActionListener(am);
        itemNero.addActionListener(am);
        itemRosso.addActionListener(am);
        itemVerde.addActionListener(am);
        itemQuadrato.addActionListener(am);
        itemCerchio.addActionListener(am);
        itemPenta.addActionListener(am);
        itemEsa.addActionListener(am);
        itemDodeca.addActionListener(am);
    }

    class ascoltaMenu implements ActionListener {
        public void actionPerformed(ActionEvent ae) {
            JMenuItem menuItem = (JMenuItem) ae.getSource();
            if (menuItem == itemBianco)   lab.setText("bianco");
            if (menuItem == itemNero)     lab.setText("Nero");
            if (menuItem == itemRosso)    lab.setText("Rosso");
            if (menuItem == itemVerde)    lab.setText("Verde");
            if (menuItem == itemQuadrato) lab.setText("Quadrato");
            if (menuItem == itemCerchio)  lab.setText("Cerchio");
            if (menuItem == itemPenta)    lab.setText("Pentagono");
            if (menuItem == itemEsa)      lab.setText("Esagono");
            if (menuItem == itemDodeca)   lab.setText("Dodecagono");
        }
    }
}
```

La figura 10.7 mostra l'applet con la barra dei menù e un menù aperto con il suo sottomenù anch'esso aperto.

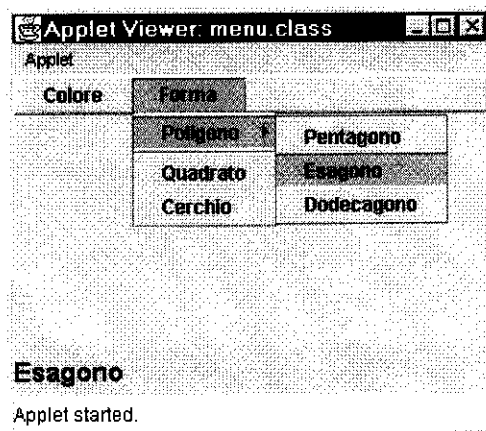



Figura 10.7 Esempio di un'applet con barra dei menù.

3.2.7 Classe **JTabbedPane**

La classe **JTabbedPane** rappresenta

Un esempio di un `JTabbedPane` è quello del programma seguente:

```
// J07tabbed.java (F.Spagna) Esempio di JTabbedPane

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;

public class J07tabbed extends JApplet {
   String[] tips = { "prima scelta", "seconda scelta", "terza scelta" };
   public void init() {
      JTabbedPane TP = new JTabbedPane(JTabbedPane.BOTTOM);
      TP.addChangeListener(new ascoltatore());
      for (int n = 0; n < 3; n++) {
         JLabel lab = new JLabel("QUESTO E' IL PANE N." + n, JLabel.CENTER);
         TP.addTab("selez." + n, null, lab, tips[n]);
      }
      getContentPane().add(TP);
   }
   class ascoltatore implements ChangeListener {
      int scelta = -1;                                     // valore iniziale
      public void stateChanged(ChangeEvent ce) {
         JTabbedPane tp = (JTabbedPane) ce.getSource();
                        // disabilita il tab selezionato e riabilita l'ex
         if (scelta == -1 || scelta != tp.getSelectedIndex()) { // se la volta
            tp.setEnabledAt(tp.getSelectedIndex(), false);
            if (scelta != -1)                              // se non prima volta
               tp.setEnabledAt(scelta, true);
```

```
        }
        scelta = tp.getSelectedIndex();          // conserva la nuova selezione
      }
    }
}
```

La figura 10.8 mostra l'aspetto dell'applet dell'esempio.

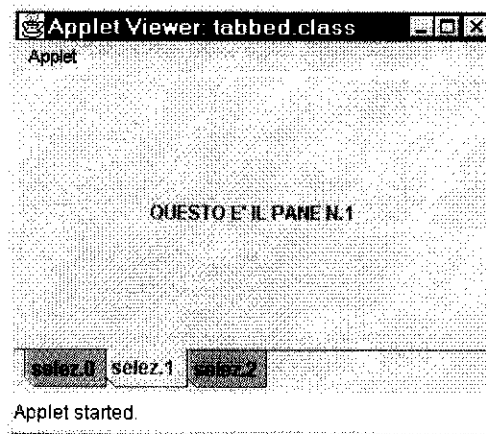



Figura 10.8 Esempio di un `JTabbedPane`.

3.2.8 Classe `JTable`

La classe **`JTable`** rappresenta

Un esempio della massima semplicità per la creazione di una tabella tipo `JTable` è il seguente:

```
// J08jtable.java (F.Spagna) Il piu' semplice esempio di JTable

import javax.swing.*;

public class J08jtable extends JApplet {

    public void init() {

        JTable tabella = new JTable(6, 5);             // 6 righe e 5 colonne
        getContentPane().add(tabella);
    }
}
```

L'esempio è assolutamente inutile, ma è stato fatto solo per mostrare la semplicità della creazione di una griglia di tipo `JTable`. In figura 10.9 è riportata la tabella come si presenta con l'appletviewer.

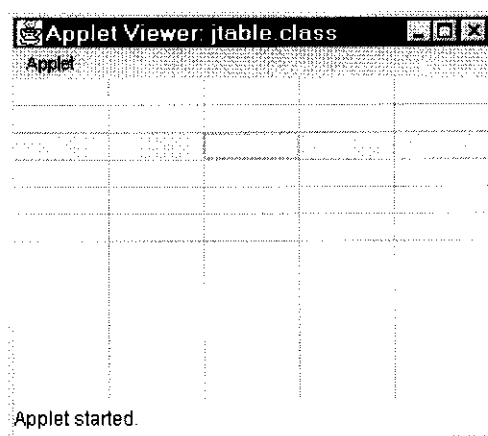



Figura 10.9 Semplice esempio di `Jtable`.

Un esempio invece con celle riempite di dati è fatto di seguito:

```
// J09jtable2.java (F.Spagna) Il piu' semplice esempio di JTable

import javax.swing.*;
import java.awt.*;
import java.util.*;

public class J09jtable2 extends JApplet {

    String[] titoli = {"italiano", "francese", "valore", "si/no"};
    Object[][] celle = {
        { "uno",     "un",     new Integer(1), new Boolean(false) },
        { "due",     "deux",   new Integer(1), new Boolean(true)  },
        { "tre",     "trois",  new Float(3.3), new Boolean(true)  },
        { "quattro", "quatre", new Float(4.4), new Boolean(false) } };

    public void init() {

        JTable tabella = new JTable(celle, titoli);
        getContentPane().add(tabella.getTableHeader(), BorderLayout.NORTH);
        getContentPane().add(tabella);
    }
}
```

In figura 10.10 si vede la griglia con le celle contenenti i dati.

Figura 10.10 Una `JTable` con celle riempite di valori.

3.2.9 Classi di testo con stile

Le classi `JTextField` e `JTextArea` presentano tutto il loro testo (rispettivamente a singola riga o a multi-righe) in uno stile unico: una volta definiti in esse un font, un colore di testo ed un colore di fondo (caratteristiche dello "stile") questi restano costanti per tutto il componente. Se si vuole invece poter assegnare uno stile diverso a diverse porzioni del testo si possono usare le due classi di componenti Swing con testo "*styled* " (o "*formatted*"), che sono la `JTextPane` e la `JEditorPane`. Facendo un confronto tra queste due classi si noti che, mentre `JEditorPane` è adatto per presentare una pagina formattata fissa, è invece carente di funzionalità per quanto riguarda l'inserimento di testo (editing), cosa per la quale `JTextPane` è molto più flessibile (anche se il suo uso non è molto semplice).

JEditorPane is used to render data content such as HTML, RTF, etc. JEditorPane is designed around the concept of a page, or a single source of data. It works very well to display fixed amounts of styled text (a Web page, for instance), but tends to lack functionality for inserting and appending text to its document once set.

JTextPane is much more malleable than JEditorPane in terms of the insertion and appending of text, but it is not quite as easy to use. In order to use a JTextPane, you need to associate an attribute set with each of the pieces of text that you insert. We'll discuss attribute sets shortly.

3.2.10 Classe **JEditorPane**

La classe **JEditorPane** permette di presentare una pagina con contenuto formattato (*styled*), per esempio in formato HTML o altro formato predefinito.

In un oggetto di `JEditorPane` ci sono due modi alternativi per mostrare una pagina: o stabilire la pagina da visibilizzare (con il metodo `setPage(URL url)`, che visualizza una pagina Web) affidandosi ad essa stessa per averne definito il tipo di contenuto, oppure stabilire esplicitamente a priori il tipo di contenuto (*content type*) per il `JEditorPane` con il metodo `setContentType(String type)` (per una pagina Web l'argomento sarebbe `"text/html"`) e poi aggiungere il testo contenuto al componente con il metodo `setPage(URL url)`, oppure il `setText(String text)`: se il testo ha già un formato HTML il `JEditorPane` si incaricherà di interpretarlo. I tag HTML riconosciuti sono quelli aderenti alla specifica HTML 3.2 (è quindi compreso anche il `<table>`).

Una cosa importante da notare è che quando sono visualizzati i file HTML, sulla pagina compaiono dei widget grafici in corrispondenza dei tag `<head>` e `<title>` del documento, a meno che si invochi il metodo `setEditable(false)` del `JEditorPane` che ha l'effetto di nasconderli.

I tag HTML riconosciuti sono quelli aderenti alla specifica HTML 3.2 (è quindi compreso anche il <table>).

Ecco qui presentato un esempio sull'uso di tale classe:

```
// J10editpane.java (F.Spagna) Esempio di JEditorPane

import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.text.*;
import java.util.*;

public class J10editpane extends JFrame{

   public editpane() {
      super("Esempio di JEditorPane");
   }
   public static void main(String[] args) {

      editpane fram = new editpane();
      JEditorPane editor = new JEditorPane();
      editor.setContentType("text/html");
      editor.setEditable(false);
      String editorText = "<html><head><title>Titolo documento</title></head>"
         + " <body bgcolor=\"white\">"
         + "<font face=\"arial, helvetica\"><div align=\"center\">"
         + "Testo centrato</div>"
         + "<table border=1 cellpadding=0 cellspacing=10><tr><td>"
         + "Cella 0,0</td><td>Cell 1,0</td>"
         + "<td>Cella 2,0</td></tr><tr><td>Cella 0,1</td>"
         + "<td>Cella 1,1</td><td>Cella 2,1</td></tr></table>"
         + "<h2>Questo è un titolo di livello 2</h2>"
```

```
            + "</font></body></html>";
        editor.setText(editorText);
        JPanel pan = new JPanel();
        pan.setLayout(new BorderLayout());
        pan.setPreferredSize(new Dimension(320, 240));
        pan.add(new JScrollPane(editor));
        fram.setContentPane(pan);
        fram.addWindowListener(new WindowAdapter() {
           public void windowClosing(WindowEvent e) {
              System.exit(0);
           }
        });
        fram.show();
        fram.pack();
    }
}
```

Il risultato è riportato nella figura 10.11 seguente.

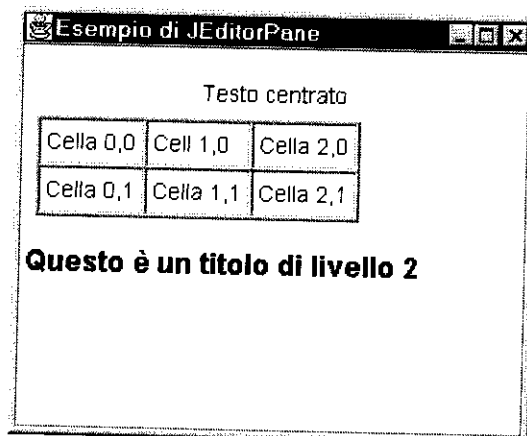



Figura 10.11 Esempio di un `JEditorPane`.

Un altro interessante esempio dell'utilizzazione di un `JEditorPane` per creare un browser Internet della massima semplicità è il seguente:

```
// J11navigatore.java (F.Spagna) Esempio di JEditorPane funzionante da browser
// tratto da un esempio di "Pure JFC Swing" [8.1]

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;

public class J11navigatore extends JFrame {
```

```
    JTextField tf;
    JEditorPane ep;

    public navigatore() {
       super("navigatore");                                     // titolo del frame
       Container cp = this.getContentPane();               // content pane del frame
       JPanel pan = new JPanel(new GridLayout(2, 1));                 // crea panel
       pan.add(new JLabel("URL richiesto: "));              // con un label messo su
       tf = new JTextField("http://");                         // crea un text field
       tf.setFont(new Font("", Font.BOLD, 14));
       tf.addActionListener(new TextFieldListener());             // con suo listener
       pan.add(tf);                                   // e lo mette quindi sul panel
       cp.add(pan, BorderLayout.NORTH);                    // panel sul content pane
       ep = new JEditorPane();                               // crea un editor pane
       ep.setEditable(false);                          // per non mostrare i widget
       JScrollPane sp = new JScrollPane(ep);                 // messo in scrollpane
       cp.add(sp);                             // e anche scroll pane sul content pane
       addWindowListener(new WindowEventHandler());  // listener chiusura frame
       setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
       setSize(450, 340);                              // caratteristiche del frame
       setBackground(Color.lightGray);
       setForeground(Color.black);
       show();                                                    // mostra il frame
       tf.requestFocus();                         // richiede il focus al text field
    }
    public static void main(String[] args) {
       new navigatore();
    }
                                         // classe interna listener del text field
    class TextFieldListener implements ActionListener {

       public void actionPerformed(ActionEvent ae) {
          URL pageURL = null;
                   try {
          pageURL = new URL(tf.getText());         // crea URL richiesto in input
                   } catch (MalformedURLException me) {
                         System.out.println("MalformedURL"); }
                   try {
          ep.setPage(pageURL);                    // pagina mostrata in editor pane
                   } catch (java.io.IOException ioe) {
                        System.out.println("IOException while loading page"); }
       }
    }
                              // classe interna listener per la chiusura del frame
     class WindowEventHandler extends WindowAdapter {

         public void windowClosing(WindowEvent evt) {
             System.exit(0);
         }
     }
}
```

Quando nel campo di input viene immessa la stringa di un'URL la pagina Web è visualizzata (con funzionalità ovviamente molto limitate). La figura 10.12 seguente fa vedere un esempio reale:

Figura 10.12 Esempio di un `JEditorPane` utilizzato come navigatore Web.

3.2.11 Classe `JTextPane`

La classe **`JTextPane`** è un'estensione della `JEditorPane` che fornisce in più le funzionalità di un vero e proprio word processor in miniatura. Con un oggetto di questa classe si può creare un testo multicolore, con diversi font e diversi stili, con possibilità di inserimento di immagini e altro. Per applicare i diversi tipi di stile al suo contenuto `JTextPane` si serve degli `AttributeSets`, che definiscono un gruppo di attributi applicabili al testo usando metodi statici della classe `StyleConstants` (un certo `AttributeSets` definito in un programma può essere riutilizzato varie volte nello stesso documento). Questi metodi consentono di fissare cose come dimensioni, colore e weight del font, come anche il rientro del rigo nell'andata a capo (*indentation*) e il "*tab spacing*".

Eccone un esempio:

```
// J12textpane.java (F.spagna) Esempio di JTextPane

import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.text.*;
import java.util.*;

public class J12textpane extends JFrame{

   public textpane() {
      super("Esempio di JTextPane");
   }
   public static void main(String[] args) {
```

```
        textpane fram = new textpane();
        JTextPane textPane = new JTextPane();
        textPane.setEditable(false);
        SimpleAttributeSet BoItR = new SimpleAttributeSet();
        StyleConstants.setForeground(BoItR, Color.red);
        StyleConstants.setBold(BoItR, true);
        StyleConstants.setItalic(BoItR, true);
        SimpleAttributeSet CeB = new SimpleAttributeSet();
        StyleConstants.setForeground(CeB, Color.black);
        StyleConstants.setAlignment(CeB, StyleConstants.ALIGN_CENTER);
        SimpleAttributeSet LarB = new SimpleAttributeSet();
        StyleConstants.setForeground(LarB, Color.black);
        StyleConstants.setFontSize(LarB, 20);
              try {
        Document doc = textPane.getDocument();
        doc.insertString(doc.getLength(), "Testo nero centrato\n", CeB);
        textPane.setParagraphAttributes(CeB, true);
        doc.insertString(doc.getLength(), "Testo bold italic rosso\n", BoItR);
        doc.insertString(doc.getLength(), "Testo nero a 20 punti\n", LarB);
              } catch(BadLocationException exp) { exp.printStackTrace(); }
        JPanel pan = new JPanel();
        pan.setLayout(new BorderLayout());
        pan.setPreferredSize(new Dimension(320, 240));
        pan.add(new JScrollPane(textPane));
        fram.setContentPane(pan);
        fram.addWindowListener(new WindowAdapter() {
           public void windowClosing(WindowEvent e) {
              System.exit(0);
           }
        });
        fram.show();
        fram.pack();
    }
}
```

La finestra risultante è raffigurata in figura 10.13.

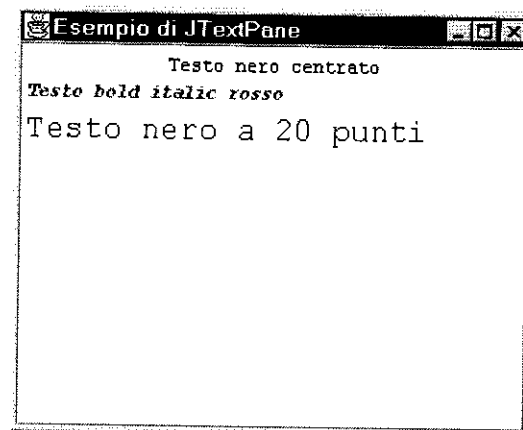



Figura 10.13 Esempio di `JTextPane`.

3.2.12 Classe **JSplitPane**

La classe **JSplitPane** rappresenta

Un esempio con uno `JSplitPane` contenente un altro `JSplitPane` è stato fatto con il codice seguente:

```
// J13split.java (F.Spagna)  Esempio di JSplitPane

import javax.swing.*;
import java.awt.*;

public class J13split extends JApplet {                       //   split pane 2
                                                               //  --------------
   boolean continuousLayout = true;                            //  | NW |       |  tre
   Icon icon1 = new ImageIcon("anjv.gif");                     //  |----|   E   |  scroll
   Icon icon2 = new ImageIcon("bann1.gif");                    //  | SW |       |  pane
   Icon icon3 = new ImageIcon("bann2.gif");                    //  --------------
                                                               //   split
   public void init() {                                        //   pane 1

      JLabel label1 = new JLabel(icon1);                          // 3 label con icona
      JLabel label2 = new JLabel(icon2);
      JLabel label3 = new JLabel(icon3);

      JScrollPane NW = new JScrollPane(label1);                    // su 3 scrollpane
      JScrollPane SW = new JScrollPane(label2);
      JScrollPane ES  = new JScrollPane(label3);

      JSplitPane splitPane1 = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
               continuousLayout, NW, SW);          // 2 scrollpane su splitpane
      splitPane1.setOneTouchExpandable(true);        // collapse/expand widget
      splitPane1.setDividerSize(2);                     // dimensione divisore
      splitPane1.setDividerLocation(0.5);   // posizione iniziale del divisore

      JSplitPane splitPane2 = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
            splitPane1, ES); // splitpane1 e terzo scrollpane su split esterno
      splitPane2.setOneTouchExpandable(true);        // collapse/expand widget
      splitPane2.setDividerSize(2);                     // dimensione divisore
      splitPane2.setDividerLocation(0.4);   // posizione iniziale del divisore

      getContentPane().add(splitPane2);  // split pane esterno su content pane
   }                                                         // dell'applet
}
```

L'applet risultante è riprodotta in figura 10.14 dopo l'intervento dell'utente.

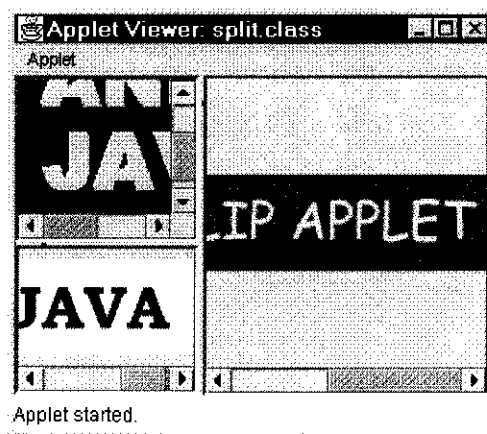



Figura 10.14 Composizione di diversi `JSplitPane`.

3.2.13 Classe **JSlider**

La classe **JSlider** rappresenta

Un esempio dell'uso di uno `JSlider` è presentato qui di seguito:

```
// J14slider.java (F.Spagna) Esempio di JSlider

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class J14slider extends JApplet {

   JLabel lab;
   JSlider slider;
   public void init() {
      Container cp = this.getContentPane();
      cp.setBackground(Color.lightGray);
      cp.setLayout(new GridLayout(4, 1));
      lab = new JLabel("Valore dello slider: 50", JLabel.CENTER);
      lab.setFont(new Font("Helvetica", Font.BOLD, 14));
      cp.add(lab);
      int min = 0, max = 100, iniz = 50;
      slider = new JSlider(JSlider.HORIZONTAL, min, max, iniz);
      slider.addChangeListener(new ascoltatore());
      slider.setPaintTicks(true);
      slider.setMajorTickSpacing(10);
      slider.setMinorTickSpacing(5);
      cp.add(slider);
      slider.setLabelTable(slider.createStandardLabels(10));
      slider.setPaintLabels(true);
   }
```

```
   class ascoltatore implements ChangeListener {
      public void stateChanged(ChangeEvent chngEvt) {
         JSlider sliderTemp = (JSlider) chngEvt.getSource();
         if (sliderTemp == slider)
            lab.setText("Valore dello slider: " + slider.getValue());
      }
   }
}
```

L'applet dell'esempio è visibile in figura 10.15.

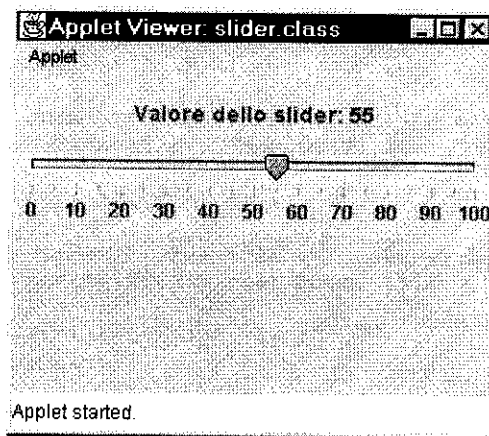



Figura 10.15 Esempio di `JSlider`.

3.2.14 Classe `Timer`

La classe **`Timer`** fornisce un mezzo per scandire dei tempi con una successione di eventi periodici.

3.2.15 Classe `JProgressBar`

La classe **`JProgressBar`** rappresenta

Ecco un esempio chge fa vedere come si può usare di `JProgressBar`:

```
// J15progress.java (F.Spagna) Esempio di Timer e JProgressBar
```

```
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;

public class J15progress extends JApplet {

   Container cp;
   JButton bot, bstop;
   JTextField tf1, tf2;
   JLabel label3;
   JProgressBar pBar;
   Timer timer;
   static int counter = 0;

   public void init() {
      cp = this.getContentPane();
      cp.setLayout(new GridLayout(6, 1));

      JLabel label1 = new JLabel(" conta fino a: ", JLabel.LEFT);
      label1.setFont(new Font("Dialog", Font.BOLD, 18));
      cp.add(label1);
      cp.add(tf1 = new JTextField("60", 4));

      JLabel label2 = new JLabel(" intervallo: ", JLabel.LEFT);
      label2.setFont(new Font("Dialog", Font.BOLD, 18));
      cp.add(label2);
      cp.add(tf2 = new JTextField("1000", 4));
      bot = new JButton("Conta");
      bot.addActionListener(new ascoltBot());
      cp.add(bot);
      bstop = new JButton("Stop");
      bstop.addActionListener(new ascoltBot());
      cp.add(bstop);
      label3 = new JLabel(" contati: 0", JLabel.CENTER);
      label3.setFont(new Font("Dialog", Font.BOLD, 18));
      cp.add(label3);
      pBar = new JProgressBar();
      pBar.setStringPainted(true);
      pBar.setBorder(BorderFactory.createLineBorder(Color.blue, 2));
      pBar.setBackground(Color.white);
      pBar.setForeground(Color.red);
      pBar.setMinimum(0);
      pBar.setMaximum(Integer.parseInt(tf1.getText()));
      cp.add(pBar);
      timer = new Timer(0, new ascoltTimer());
   }

   class ascoltTimer implements ActionListener {

      public void actionPerformed(ActionEvent e) {
         counter++;
         pBar.setValue(counter);
         label3.setText(" contati: " + counter);
         if (counter >= Integer.parseInt(tf1.getText()))
            timer.stop();
      }
   }

   class ascoltBot implements ActionListener {

      public void actionPerformed(ActionEvent e) {
```

```
        JButton button = (JButton) e.getSource();
        if (button.getText() == "Conta") {
           timer.setDelay(Integer.parseInt(tf2.getText()));
           label3.setText(" contati: 0");
           pBar.setMaximum(Integer.parseInt(tf1.getText()));
           counter = 0;
           timer.start();
        }
        if (button.getText() == "Stop") {
           counter = 0;
           timer.stop();
        }
     }
   }
}
```

La figura 10.16 mostra l'applet in fase di funzionamento.

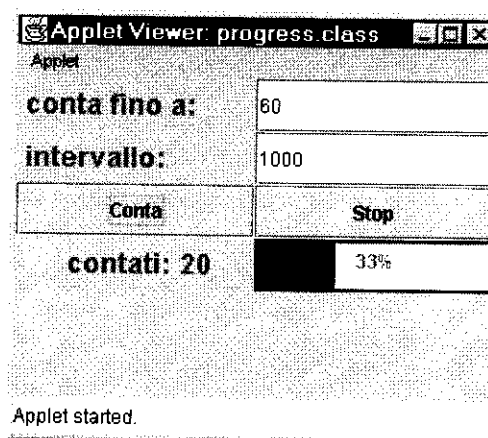



Figura 10.16 Esempio di `JProgressBar` e di `Timer`.

3.2.16 Classe `JFileChooser`

La classe **`JFileChooser`** permette la creazione di

Ecco un esempio di come si usa `JFileChooser` qui di seguito:

```
// J16chooser.java (F.Spagna) Esempio di JFileChooser

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
```

```
import java.awt.event.*;
import java.io.*;

public class J16chooser extends JFrame {

   JMenuItem openMI, saveMI, saveAsMI, exitMI;
   JFileChooser fc;
   Container cp;
   JLabel lab;
   File file, fileScelto;

   public chooser() {
      this.addWindowListener(new FrameClosing());
      cp = this.getContentPane();
      lab = new JLabel();                          // per messaggi vari sul fondo
      lab.setFont(new Font("", Font.BOLD, 20));
      cp.add(lab);
      JMenuBar menuBar = new JMenuBar();
      menuBar.setBorder(BorderFactory.createEtchedBorder());
      cp.add(menuBar, BorderLayout.NORTH);
      JMenu menuFile = new JMenu("File");
      menuBar.add(menuFile);
      JMenuItem newMI = new JMenuItem("Nuovo");
      newMI.setEnabled(false);
      menuFile.add(newMI);
      openMI = new JMenuItem("Apri");
      openMI.addActionListener(new MIActionListener());
      menuFile.add(openMI);
      menuFile.addSeparator();
      saveMI = new JMenuItem("Salva");
      saveMI.setEnabled(false);
      menuFile.add(saveMI);
      saveAsMI = new JMenuItem("Salva con nome");
      saveAsMI.addActionListener(new MIActionListener());
      menuFile.add(saveAsMI);
      menuFile.addSeparator();
      menuFile.addSeparator();
      exitMI = new JMenuItem("Esci");
      exitMI.addActionListener(new MIActionListener());
      menuFile.add(exitMI);
      JMenu editMenu = new JMenu("Modifica");
      menuBar.add(editMenu);
   }
   class MIActionListener implements ActionListener {
      public void actionPerformed(ActionEvent ae) {
         JMenuItem menuItem = (JMenuItem) ae.getSource();
         if (menuItem == openMI) {
            if (file == null)
               fc = new JFileChooser();          // punta sulla home directory
            else
               fc = new JFileChooser(file);  // punta su file path specificato
            int selected = fc.showOpenDialog(cp);
            if (selected == JFileChooser.APPROVE_OPTION) {     // bottone Open
               file = fc.getSelectedFile();
               lab.setText("File selezionato da aprire: "+ file.getName());
               lab.setHorizontalAlignment(JLabel.CENTER);
               return;
            }
            else if (selected == fc.CANCEL_OPTION) {
               lab.setText("Non selezionato file da aprire");
               lab.setHorizontalAlignment(JLabel.CENTER);
               return;
```

```
            }
        }
        else if (menuItem == saveAsMI) {
           // inserire qui il codice per salvare il file...
           fc = new JFileChooser();
           int selected = fc.showSaveDialog(cp);
           fileScelto = new File("UNTITLED");
           fc.setSelectedFile(fileScelto);
           if (selected == JFileChooser.APPROVE_OPTION) {
              fileScelto = fc.getSelectedFile();
              lab.setText("File selezionato da salvare: "+ file.getName());
              lab.setHorizontalAlignment(JLabel.CENTER);
              return;
           }
           else if (selected == fc.CANCEL_OPTION) {
              lab.setText("Non selezionato file da salvare");
              lab.setHorizontalAlignment(JLabel.CENTER);
              return;
           }
        }
        else if (menuItem == exitMI)
           System.exit(0);
     }
  }
  class FrameClosing extends WindowAdapter {
     public void windowClosing(WindowEvent e) {
        System.exit(0);
     }
  }
  public static void main(String[] args) {
     chooser frame = new chooser();
     frame.setTitle("Esempio di JFileChooser");
     frame.setSize(450, 300);
     frame.setVisible(true);
  }
}
```

Le figure 10.17, 10.18 e 10.19 mostrano l'applicazione rispettivamente nello stato iniziale e all'apertura delle finestre di apertura file e salvataggio file.

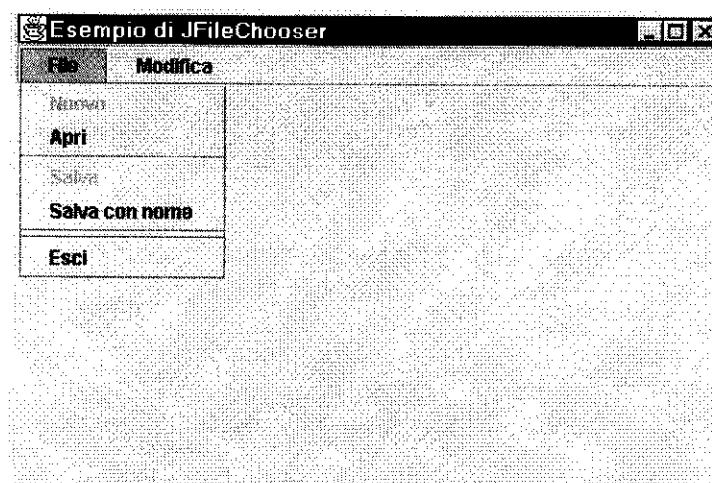



Figura 10.17 Applicazione con un `JFileChooser` allo stato iniziale.

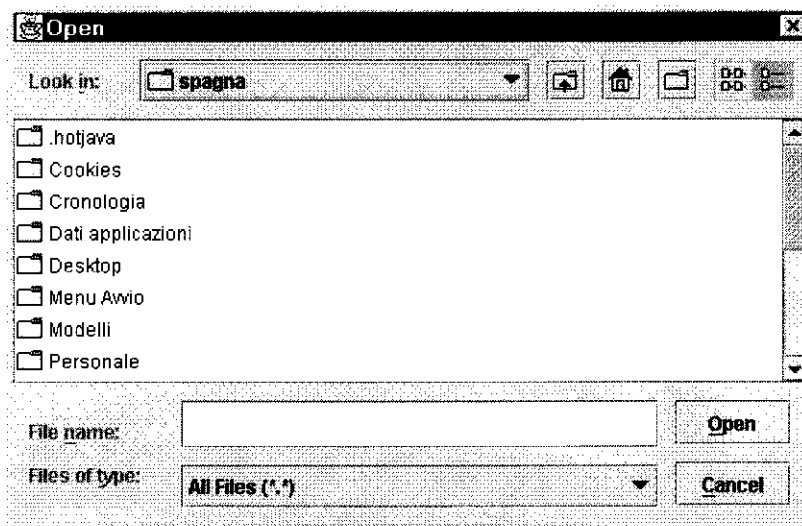



Figura 10.18 Finestra di apertura file in un'applicazione con un `JFileChooser`.

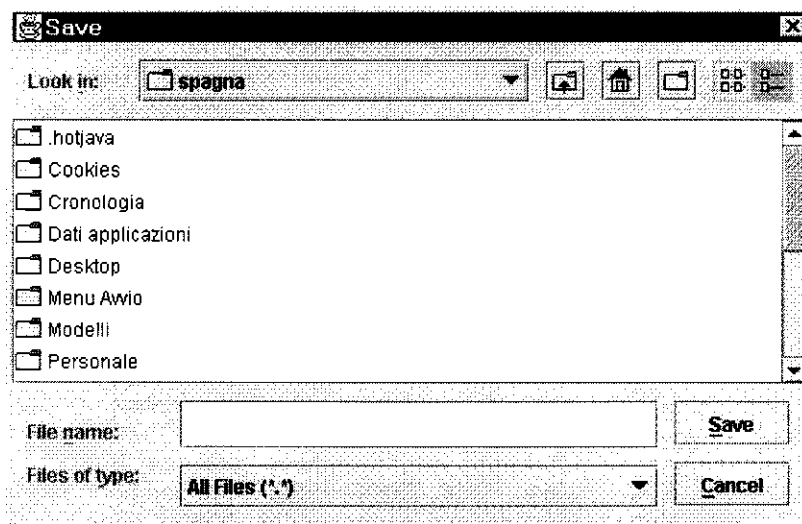



Figura 10.19 Finestra di salvataggio file in un'applicazione con un `JFileChooser`.

3.2.17 Classe **JPopupMenu**

La classe **JPopupMenu** permette il richiamo di un menù tipo popup al clic del bottone destro del mouse su un componente.

Presentiamo un esempio di `JPopupMenu`:

```
// J17popup.java (F.Spagna) Esempio di JPopupMenu

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class J17popup extends JFrame {

   JPopupMenu popupMenu;
   JMenuItem item[] = new JMenuItem[5];
   String nome[] = { "uno", "due", "tre", "quattro", "cinque" };
   Container container;

   public popup() {
      this.addWindowListener(new FrameClosing());
      container = this.getContentPane();
      JLabel lab = new JLabel("Fare click con bottone destro del mouse");
      lab.setFont(new Font("", Font.BOLD, 16));
      container.add(lab);
      popupMenu = new JPopupMenu("Test Popup Menu");
      for (int n = 0; n < 5; n++) {
         item[n] = new JMenuItem(nome[n]);
         popupMenu.add(item[n]);
         if (n==0 || n==3)
            popupMenu.addSeparator();
      }
      PopupMenuListener pml = new PopupMenuListener();
      container.addMouseListener(pml);
   }
   class PopupMenuListener extends MouseAdapter {
      public void mousePressed(MouseEvent me) {
         showPopup(me);
      }
      public void mouseReleased(MouseEvent me) {
         showPopup(me);
      }
      private void showPopup(MouseEvent me) {
         if (me.isPopupTrigger()) {
            popupMenu.show(me.getComponent(), me.getX(), me.getY());
         }
      }
   }
   class FrameClosing extends WindowAdapter {
      public void windowClosing(WindowEvent e) {
         System.exit(0);
      }
   }
   public static void main(String[] args) {
      popup frame = new popup();
      frame.setTitle("JPopupMenu");
```

```
        frame.setSize(350, 150);
        frame.setVisible(true);
    }
}
```

La figura 10.20 mostra l'applicazione con il menù attivato con un clic del tasto destro del mouse in un punto in alto a destra del contenitore.

Figura 10.20 Esempio di `JPopupMenu`.

SOMMARIO